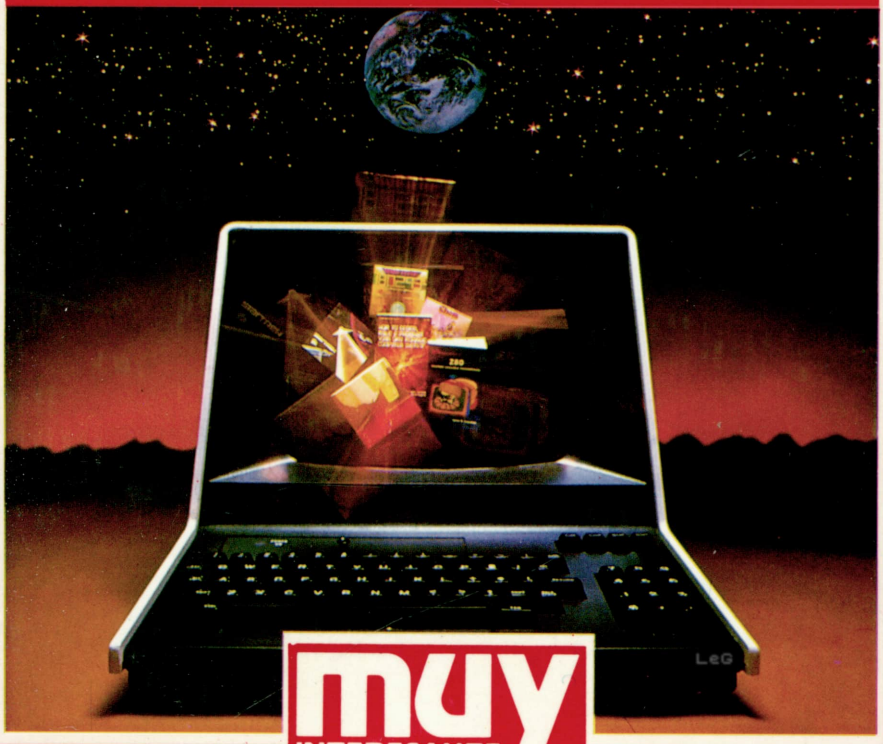
LIBRO BÁSICO DEL ORDENADOR PERSONAL (II)

3. Hardware y software

Peter Rodwell



muy INTERESANTE

BIBLIOTECA DE DIVULGACIÓN CIENTÍFICA

Biblioteca
de Divulgación Científica

LIBRO BASICO DEL ORDENADOR PERSONAL

PETER RODWELL

II

3. Hardware y software

Ediciones Orbis, S.A.

LIBRO BASICO DEL ORDENADOR PERSONAL

Título original:
The Personal Computer Handbook
Publicado por
Dorling Kindersley Ltd., London

Project Editor:
Jonathan Hilton
Art Editor:
Neville Graham
Editor:
Richard Dawes
Designer:
Mark Richards
Managing Editor:
Alan Buckingham
Art Director:
Stuart Jackman

Traducción:
Remigio Gómez
Asesor científico de la colección:
Pedro Puigdomènech
Director editorial:
Virgilio Ortega

© Dorling Kindersley Ltd.,
London, 1983, 1984.
© Peter Rodwell, 1983, 1984,
del texto original.
© MicroTextos, s.a. de ediciones,
Madrid, 1984 de la 1ª edición
y de la traducción en español.
© Ediciones Orbis, S.A.
Barcelona, 1986
por la presente edición
© por la ilustración de la cubierta:
Michel Tchererkoff/The Image Bank

Nota del editor:
El *Libro básico del ordenador personal* será publicado en tres volúmenes, que tendrán los números 67, 71 y 75 de esta colección.

ISBN (obra completa): 84-7634-646-8
ISBN (libro 71): 84-7634-648-4
Depósito Legal: NA-699-1986 (II)

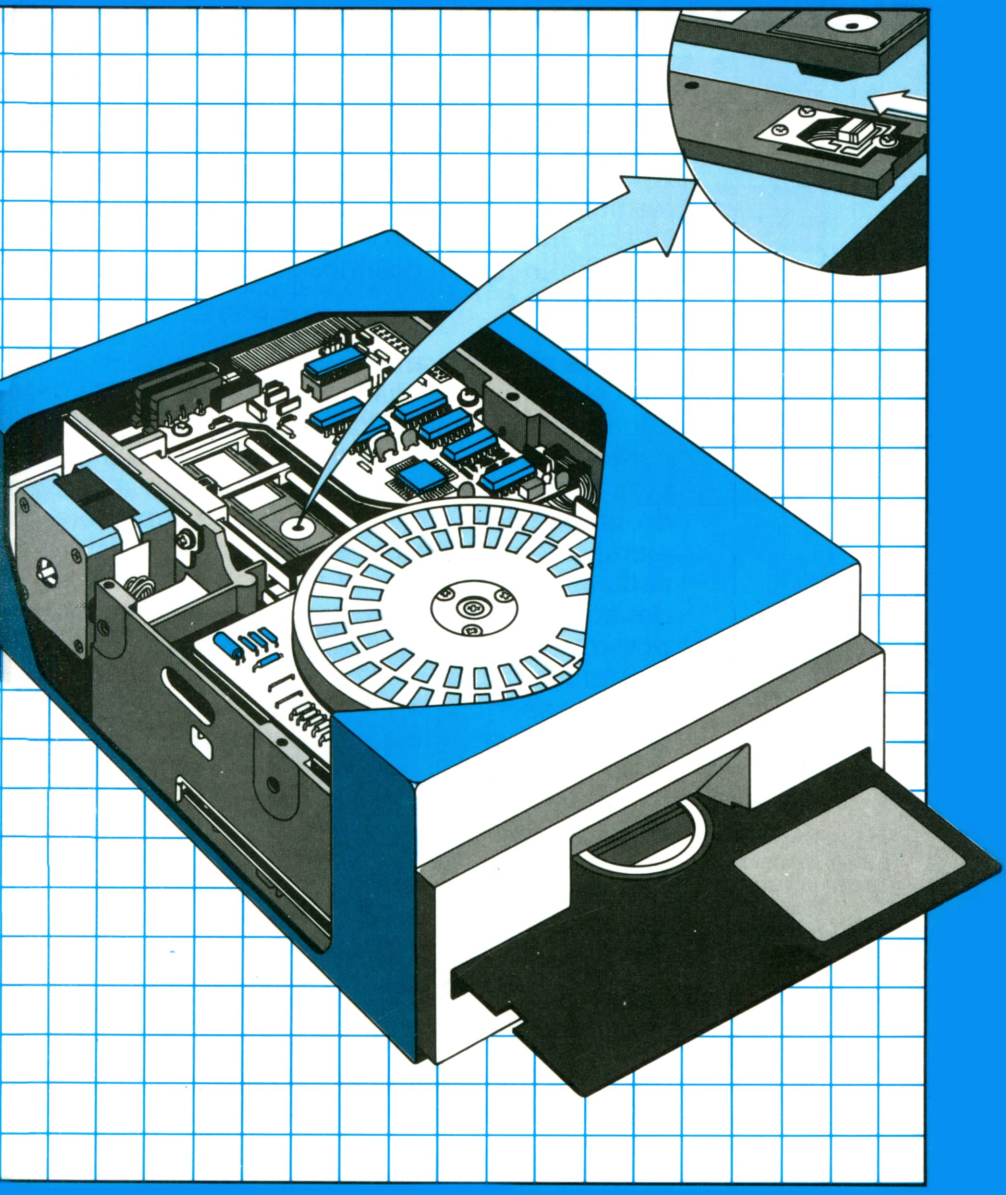
Fotocomposición:
Pérez-Díaz, S.A., Madrid

Fotomecánica:
F. E. Burman Ltd., London
Progreso Gráfico, Madrid

Impresión y encuadernación:
Gráficas Estella, S.A.
Estella (Navarra), 1986

Printed in Spain

HARDWARE



EL HARDWARE Y LA CPU

Nadie sabe con certeza dónde se originaron los términos *hardware* y *software*, pero, según va siendo más conocido y estudiado el ordenador, parecen ir popularizándose fuera de los círculos informáticos.

¿Qué es el *hardware*?

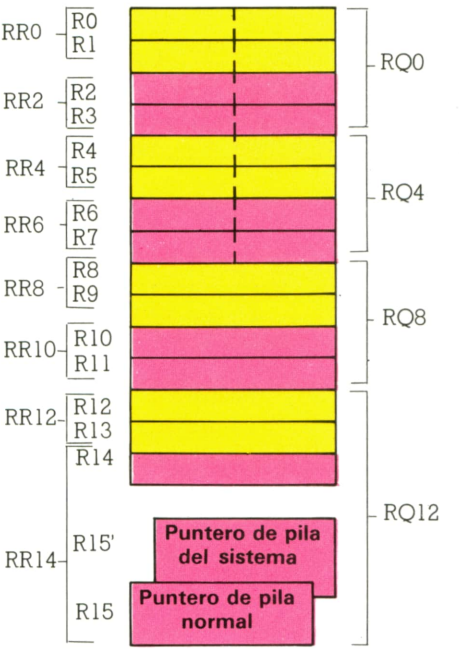
Con el nombre de *hardware* se designa el ordenador mismo y todos los demás dispositivos electrónicos y mecánicos que pueden conectarse a él para constituir el sistema completo: pantalla, teclado, unidad de disco, magnetófono e impresora, por ejemplo. Sin embargo, el *hardware* no sirve de nada sin el *software*. Si vamos a comprar un ordenador, sobre todo si es para gestión, lo más importante es el *software* (véase Módulo 25). Si lo queremos para hacer un trabajo concreto, lo primero es asegurarse de contar con el *software* (o con la persona que sepa escribirlo). Sólo una vez resuelto este tema procede empezar a buscar el *hardware* más adecuado para soportar ese *software*. El tipo de *software* que puede ejecutarse en un ordenador concreto, y su funcionamiento, dependen de la CPU.

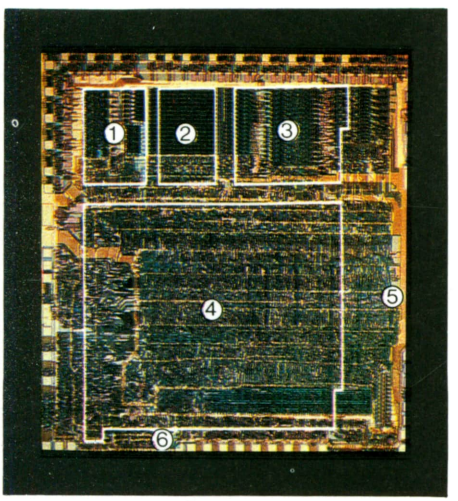
La CPU

El corazón de todo sistema informático es la CPU o Unidad Central de Proceso *(Central Processing Unit)*. Es el «cerebro» del ordenador, la parte que realiza el trabajo más difícil. En los grandes ordenadores, la CPU es un gabinete del tamaño de un

EL Z80 Y EL Z8000 DE ZILOG

El Z80 de Zilog de 8 bits (páginas 158-9) es uno de los microprocesadores más difundidos, y desciende del Intel 8080, base del sistema operativo CP/M (Módulo 33). El Z80 cuenta con un gran número de «registros» —celdas de memoria RAM—, que permiten al programador hacer muchas cosas más deprisa. El Z8000 (abajo) es el primer procesador de 16 bits de Zilog. En lugar de usar código de máquina para convertir las instrucciones binarias que le llegan, está «cableado» *(hardwired)*, es decir, los circuitos lógicos traducen directamente las instrucciones.





«Arquitectura» interna. Partes importantes del Z8000 son: (1) banderas, (2) la ALU, (3) registros, (4) circuitos lógicos, (5) y (6) controles de canal. Estas zonas controlan los datos entre la CPU y el resto del sistema. En el diagrama de los registros (arriba) se aprecia la distribución del Z8002. Los de 16 bits (del R0 al R14) pueden enlazarse por parejas (del RR0 al RR14) para manejar números de 32 bits.

Juego principal de registros

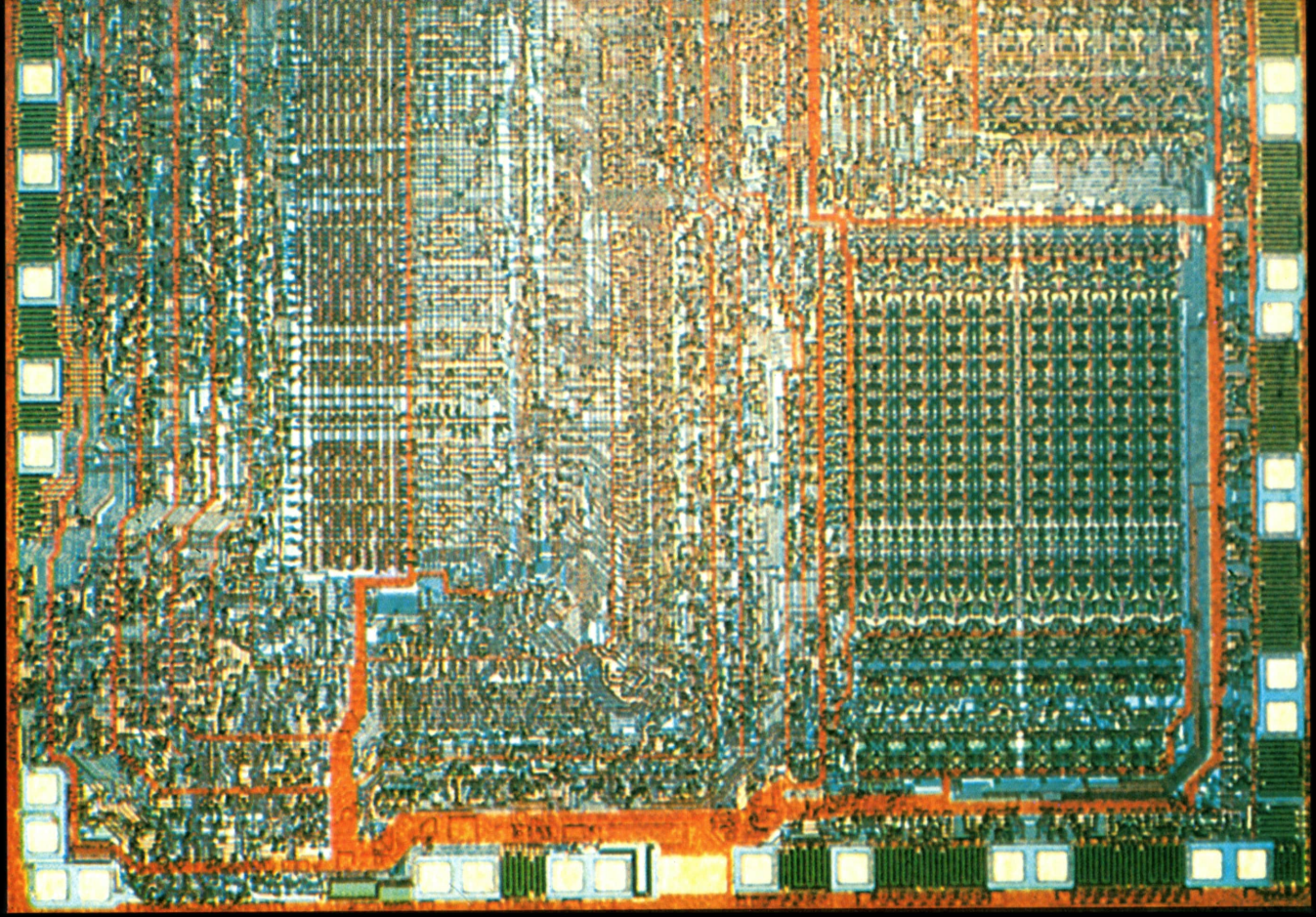
	Acumulador	Banderas
Registros generales	B	C
	D	E
	H	L

Juego secundario de registros

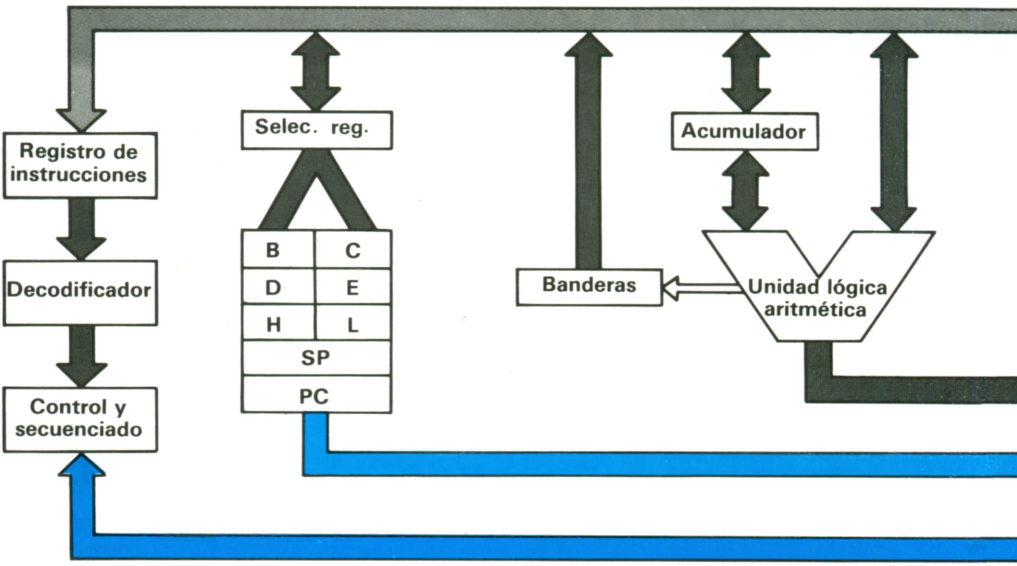
	Acumulador	Banderas
Registros generales	B	C
	D	E
	H	L

Registros especiales	Registro de índice X	
	Registro de índice Y	
	Contador de programa	
	Puntero de pila	
	Vector de interrupción	Refresco de memoria

Registros del Z80. Mostramos aquí la distribución de los registros del Z80. Los registros se nombran con designaciones arbitrarias (B, C, D, E, H, L), existiendo un juego de registros secundario que los duplica para ganar en velocidad y potencia. Las banderas indican los resultados de varias operaciones que tienen lugar en la ALU.



ESQUEMA DE CPU



Las tres unidades de la izquierda reciben instrucciones de la memoria del ordenador a través del *bus* de datos y las convierten en señales que sirven para activar distintas partes de la CPU. Los registros (B, C, D, E, H, L) son pequeñas zonas de RAM que contienen provisionalmente los datos que necesita la CPU. El contador de programa (PC) envía a través del *bus* de direcciones la posición en memoria principal de la siguiente instrucción que precisa la CPU. En el registro de banderas hay varios elementos de un solo bit, cuyo valor se ve afectado por las operaciones que tienen lugar en la ALU. En el acumulador se almacenan los datos antes y después de ser procesados por la ALU.

armario abarrotado de componentes electrónicos; en un micro personal es un único componente, llamado microprocesador.

En la foto de las páginas 158-159 vemos un microprocesador antes de encapsularlo en su envoltura de plástico protector. A primera vista parece una maraña de líneas sin sentido, pero mirándolo bien se aprecian ciertos rasgos. En primer lugar, fijémonos en los cuadraditos que hay alrededor del *chip:* se trata de puntos de conexión, y antes de envolver el *chip* en plástico se sueldan a ellos unos finos hilos de oro para conectarlos a las patas del soporte externo. Por ellos se transmiten al resto del sistema datos, direcciones de memoria y señales de control.

***Bus* de datos.** Conduce las instrucciones y la información a/y desde la memoria.

Las diversas zonas del *chip* realizan funciones diferentes. Hay una zona con esquemas geométricos bastante regulares. Es la zona de la ROM, y forma parte de la unidad de control —esta unidad traduce las instrucciones que proceden del «*bus* de datos» en señales que reconfiguran todas las puertas lógicas para posibilitar la ejecución de tales instrucciones. En el corazón del funcionamiento de la CPU está la ALU (véase Módulo 14), que es la que realiza los cálculos. En los «registros», compuestos cada uno de ellos por unos pocos octetos de RAM, se almacenan datos que precisa la ALU para ejecutar sus instrucciones.

***Bus* de direcciones.** Especifica la posición de memoria requerida.

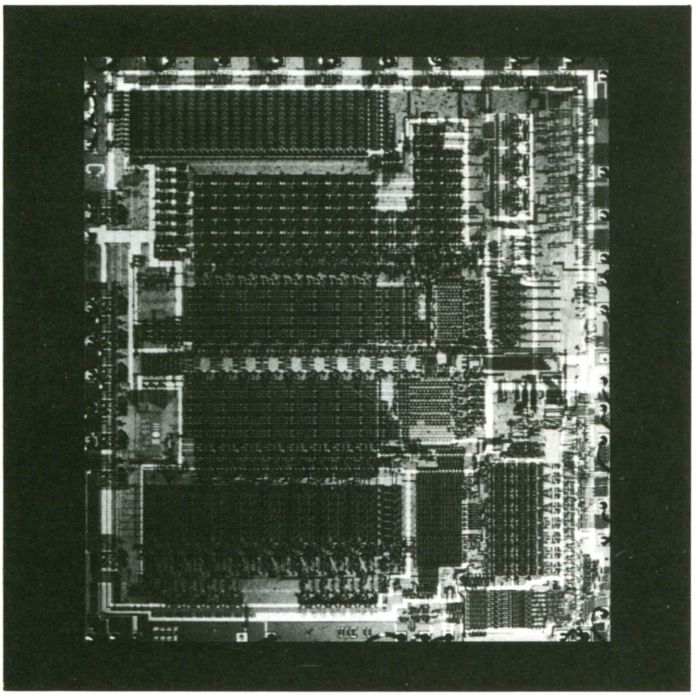
***Bus* de control.** Porta señales que describen la operación en curso.

Cómo funciona la CPU

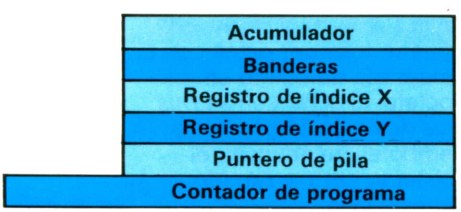
Para comprender mejor cómo funciona la CPU conviene seguir la marcha de una instrucción por el sistema, observando los sucesivos acontecimientos. Supongamos que queremos extraer dos números de la memoria, sumarlos y poner el resultado en una tercera posición de memoria. El esquema de la izquierda explica esquemáticamente la conexión de las distintas partes de la CPU. Las conexiones se llaman «*buses*» y transportan por el *chip* instrucciones y datos. Aunque la explicación de lo que ocurre en el interior de la CPU

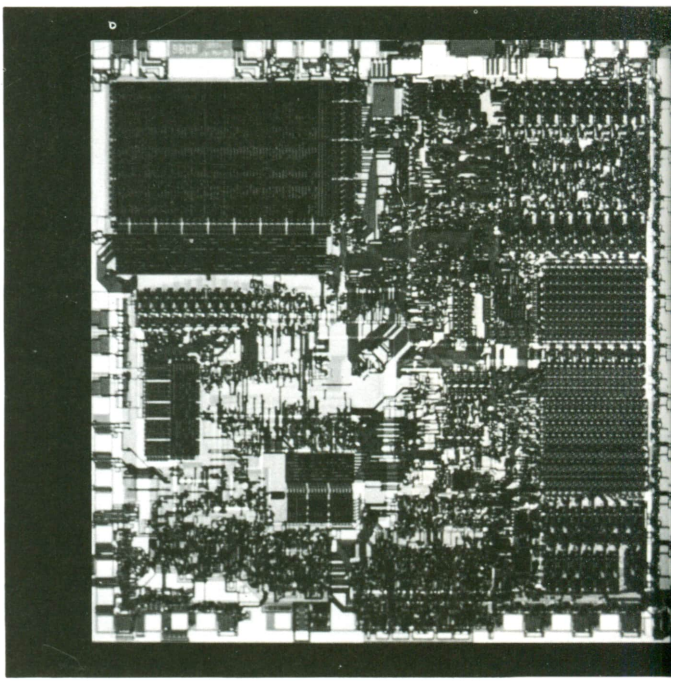
pueda parecer complicada, se trata en realidad de una descripción simplificada. Por ejemplo, la CPU no puede ejecutar por sí misma estas instrucciones; ha de haber un programa almacenado en memoria que le diga lo que tiene que hacer. En cada fase, la CPU debe recibir la siguiente instrucción del programa, y por ello se utiliza un registro especial llamado contador de programa, que indica en qué posición de memoria debe buscar el ordenador la siguiente instrucción. El contador de programa se inicializa con el valor correspondiente a la posición de memoria que contiene el primer paso del programa. Cada vez que se necesita una nueva instrucción se incrementa el contador y se envía su contenido por el *bus* de direcciones para solicitar a los *chips* de memoria que manden la instrucción contenida en esa determinada posición a la CPU (por el *bus* de datos). El contenido del contador de programa puede modificarse incluyendo en el programa la instrucción adecuada, siendo ésta una de las propiedades clave de los ordenadores: es posible alterar el orden en que se recorren las instrucciones del programa *desde el propio programa.* Por regla general esto ocurre como consecuencia de una condición derivada de una operación previa. Por ejemplo, si el resultado de nuestra suma es mayor que alguna cantidad, podríamos saltar a otra región de la memoria y ejecutar las instrucciones que se encuentren en ella. Si no lo fuese, continuaríamos con la instrucción siguiente.

MOS TECH 6502

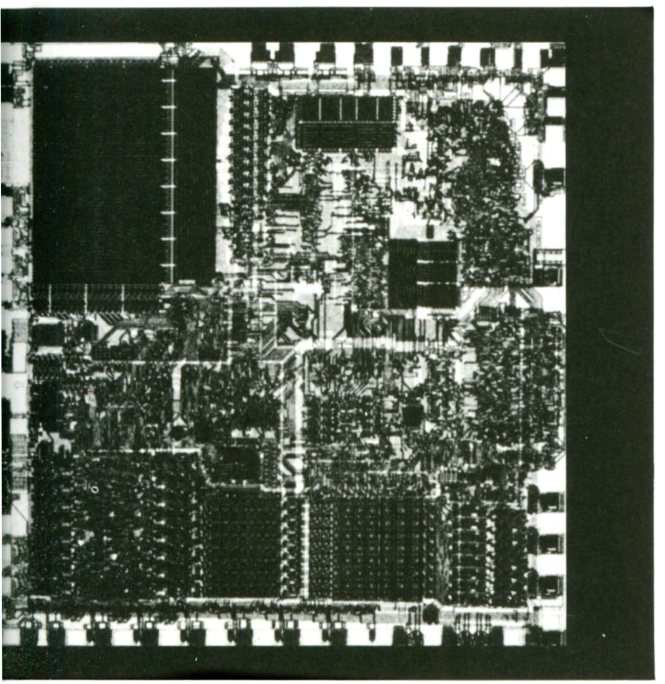


El 6502 es un popular procesador de 8 bits usado en Commodore, Apple y Atari. Aunque no sea tan potente como el Z80, no deja de poseer características muy útiles para la programación. Se han fabricado distintas versiones del 6502, que para unos suponen mejoras y para otros complicaciones. Esto hace que el *chip* acceda mucho a memoria y opere con relativa lentitud. Algunas variantes del Z80 llegan a ser tres veces más rápidas que el 6502. El juego de instrucciones del 6502 es totalmente diferente al del Z80, y por tanto no puede usar el *software* de éste.





Internamente, estos procesadores de 16 bits son casi idénticos, con los mismos registros y juegos de instrucciones. El 8088 (a la derecha) se conecta al exterior a través de un *bus* de datos de 8 bits. Esto significa que sólo puede trasegar datos con la memoria de octeto en octeto. El 8086, con su *bus* de datos de 16 bits, evita embotellamientos en el sistema, pero, a cambio, cuesta más.



Acumulador	AH	AL
Base	BH	BL
Contador	CH	CL
Dato	DH	DL

Puntero de pila
Puntero de base
Indice de origen
Indice de destino

Segmento de código
Segmento de dato
Segmento de pila
Segmento suplementario

Puntero de instrucciones	
Banderas H	Banderas L

EL CHIP

A veces se habla de los *chips* sin explicar lo que son ni cómo se fabrican. Externamente un *chip* es un bloque de plástico negro con dos hileras de patas, pero el verdadero *chip* está dentro, no es visible, salvo en el caso de las EPROM (véase Módulo 19), que cuentan con una ventana para que se pueda borrar el contenido con luz ultravioleta. Incluso en la EPROM lo único que se ve es un cuadradito plateado que lleva grabados en su superficie unos trazos diminutos.

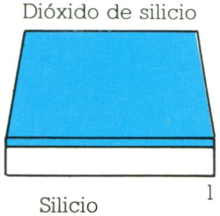
Fabricación del *chip*

La fabricación del *chip* comienza con un diseño lógico. Este diseño reviste la forma de una representación diagramática de todas las «puertas» necesarias y de las conexiones precisas para que el *chip* haga lo que queremos. Esencialmente, una puerta es un circuito digital que sólo funciona cuando recibe cierto tipo de entrada. En los *chips* complejos se realiza el diseño interno con ayuda de ordenadores. Las puertas están constituidas por las uniones de capas de silicio sometidas a diferentes tratamientos. Posteriormente hay que traducir el diagrama lógico a una serie de «máscaras», que podemos considerar una especie de negativos fotográficos, cada una de las cuales representa una de estas capas. También aquí se utilizan técnicas de diseño asistido por ordenador. El diseño de las máscaras se hace con dibujos grandes que luego se reducen fotográficamente al tamaño del *chip* acabado.

LA FABRICACION DE *CHIPS*

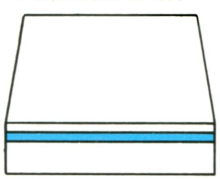
Comienza con un disco de silicio purísimo muy pulido. Como la más leve mota de polvo puede estropear el *chip*, todo el proceso de fabricación se lleva a cabo en una atmósfera muchísimo más limpia que la del quirófano de un hospital. Todo el personal lleva gorro, bata y máscara para evitar la contaminación.

Se calienta la oblea de silicio en una atmósfera de oxígeno puro para que se forme en su superfie una capa de dióxido de silicio.



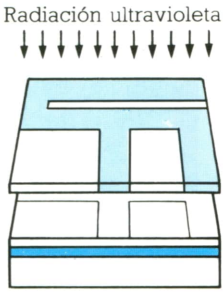
1

Luego se recubre la superficie con fotoendurecible. Esta sustancia se vuelve resistente a la acción de ciertos disolventes.



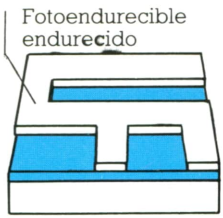
2

Se expone la oblea a la luz ultravioleta, filtrada por una máscara de película fotográfica con el dibujo de las líneas conductoras. Un lavado elimina el endurecible quedando en la oblea una copia del dibujo de la máscara.



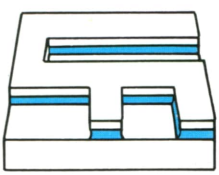
3

Un baño ácido se come todo el dióxido de silicio descubierto por la eliminación del endurecible.

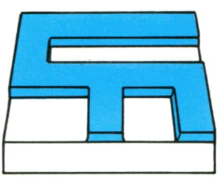


4

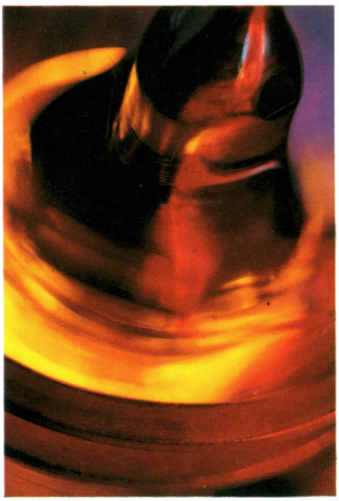
A otras capas se las «dopa» para modificar sus propiedades eléctricas. Otro lavado da lugar a nuevas conexiones.


5

Por último, se deposita una capa de aluminio sobre la superficie de silicio para conectar distintas partes de cada *chip*. Luego se desprende cada *chip* y se verifica.


6

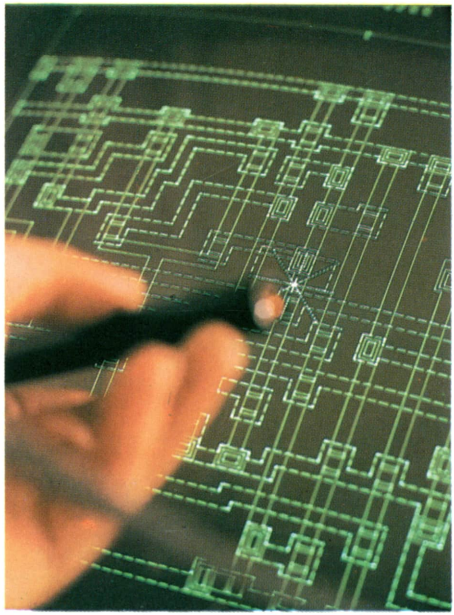
DEL SILICIO AL MICROCHIP



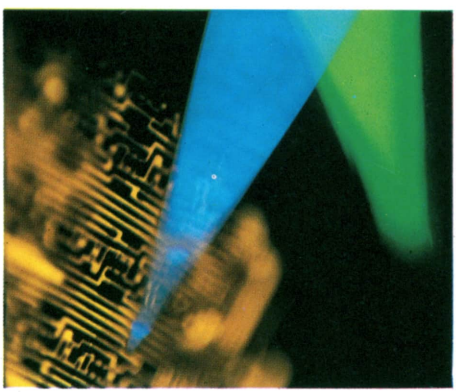
1

La fabricación de un *chip* comienza con el silicio en bruto hasta conseguir un cristal con forma de «salchicha» (1). Tras un cuidado mecanizado, se corta el cristal en láminas llamadas obleas *(wafers)* (2), sobre las que se irán construyendo los *chips*, capa a capa, utilizando un proceso de principios análogos a la impresión fotográfica.

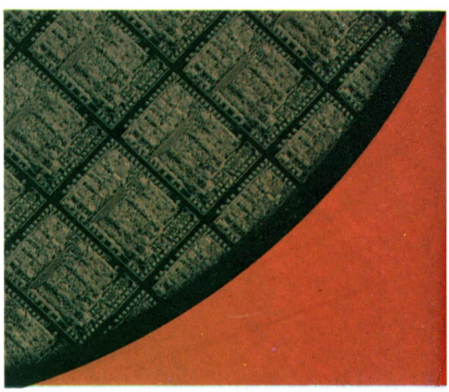
2



3



4



5

Como los *chips* actuales son muy complejos, se utilizan abundantemente técnicas de diseño asistido por computador, tanto para el diseño lógico como para realizar el diseño de las máscaras litográficas utilizadas para exponer cada una de las capas del *chip* (3). Estas máscaras son tan enrevesadas que hay que usar haces de electrones para exponer el silicio a la máscara (4), pues la finura de las líneas hace imposible que la luz visible las atraviese.
Tras una compleja elaboración queda formado un grupo de *chips* sobre cada oblea (5).

En la fabricación de *chips* se prepara una serie de cada una de las capas de máscaras y se van montando un gran número de *chips* simultáneamente en una misma oblea de silicio. El proceso es complicado, siendo preciso tratar el silicio con productos químicos fotosensibles, exponerlo a la luz a través de cada una de las máscaras, por turno, con etapas intermedias para lavar las partes no expuestas de la superficie de la oblea, y repetir el proceso hasta que todos los *chips* están terminados (véanse páginas 169-170).

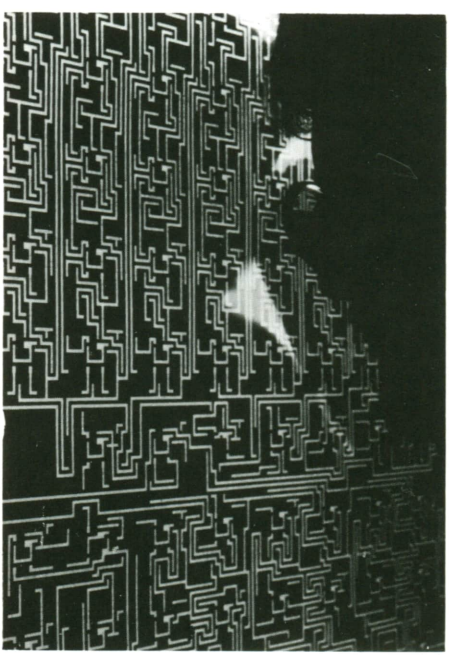
Hay una lucha constante por meter más y más puertas en el mismo *chip,* y los fabricantes se ven obligados a utilizar técnicas en las que intervienen haces electrónicos en lugar de luz, pues tal es la finura del diseño que las líneas son más estrechas que la longitud de onda de la luz.

Una solución aparentemente obvia para este problema de espacio sería usar trozos de silicio de mayor tamaño para los diseños de los *chips* más complejos, pero esto no funciona. Existe un tamaño de *chip* óptimo; si lo aumentamos, se fabricarán menos *chips* de un golpe, con el consiguiente aumento de precio, mientras que si no lo alcanzamos quedará menos espacio para colocar puertas lógicas.

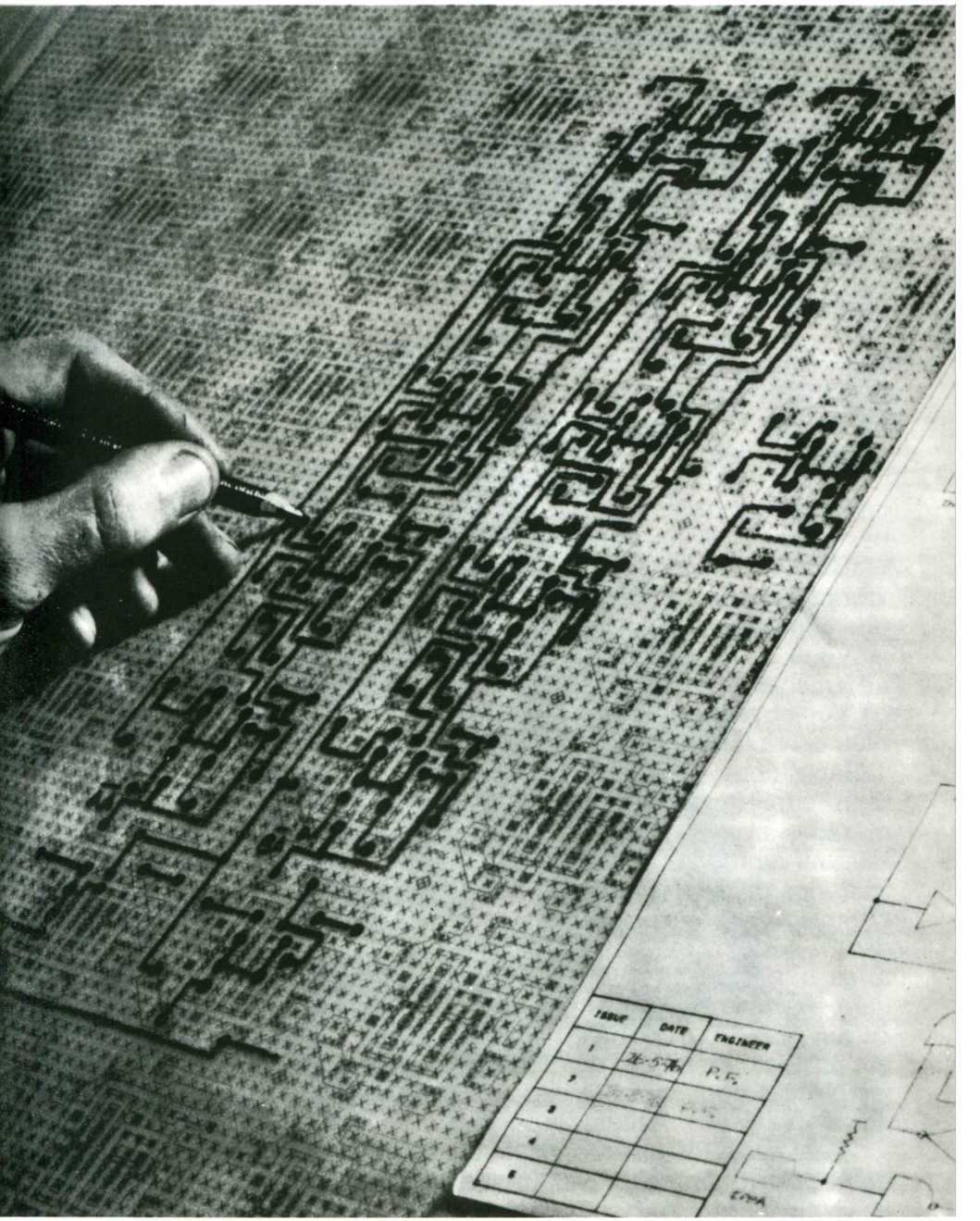
Después se parten las grandes obleas de silicio en las que se fabrican simultáneamente docenas de *chips* y se comprueban individualmente. Cada *chip* se monta en una máquina informatizada especial para comprobar que lleva a cabo correctamente sus funciones electrónicas. En estas primeras fases habrá muchos *chips* que no superen la prueba. La tasa de rechazo es un factor importante en la fabricación de *chips,* y explica en parte la reducción del precio de un *chip* nuevo, especialmente una CPU compleja, a medida que el porcentaje de *chips* válidos producidos por fábrica va aumentando.

Los *chips* que superan las pruebas son conectados a una serie de patas con hilos finos de oro y empotrados en envolturas protectoras de plás-

tico o cerámica. Se somete a una última prueba a los *chips* ya terminados antes de servirlos a los clientes. No deja de ser curioso que resulte imposible comprobar todos los aspectos de un *chip* complejo: en recorrer todo el juego de instrucciones del *chip* (con todas las combinaciones de datos posibles) tardaría un ordenador varios años.



Diseño de *chips*. Este proceso comienza con un diagrama lógico (véase la figura de la derecha), que se traduce luego en unos trazos que componen las máscaras a partir de las cuales se fabrica el *chip*. Estas máscaras se realizan a un tamaño mucho mayor que el *chip* definitivo, para conseguir la mayor precisión posible. Tras detalladas comprobaciones (véase izquierda), se copian fotográficamente y se reducen al tamaño del *chip* definitivo.


ISSUE
DATE
ENGINEER
P.E.

LA MEMORIA

Como veremos en el Módulo 27, el ordenador no hace nada si no se le da una lista de instrucciones en código binario que le diga paso por paso lo que tiene que hacer exactamente.

Para que el ordenador lo pueda leer y ejecutar, un programa debe estar almacenado en forma accesible para él. En otras palabras, es preciso enviar los unos y ceros a la CPU a través de conexiones eléctricas (véase Módulo 17). Es necesario por tanto tener el programa en una memoria. La mayoría de los programas manejan algún tipo de información, sea tecleada por el usuario, generada por el propio programa, o ambas cosas. También estos datos han de estar en memoria.

Cabe considerar la memoria como un gran muro cubierto de celdillas. Las celdillas están numeradas a partir de cero y en cada una puede haber una «unidad» de información (en la mayor parte de los microordenadores es un octeto). Imaginemos que cada celdilla está conectada a través de una serie de puertas lógicas (véase Módulo 14) a los *buses* de direcciones y datos del sistema. Para examinar el contenido de una determinada celdilla, la CPU deposita en el *bus* de direcciones el número o dirección de dicha celdilla. Las puertas lógicas conectadas a cada celdilla harán que una copia de su contenido pase al *bus* de datos, y de ahí a la CPU, cuando y sólo cuando aparezca en el *bus* de direcciones la dirección de la celdilla. Nótese que a la CPU le llega una copia.

EL *CHIP* DE MEMORIA EPROM

Las EPROM *(Erasable Programmable ROM)* difieren de las ROM en que se puede modificar su contenido. El *chip* de EPROM tiene una pequeña ventana que permite exponerlo a una intensa luz ultravioleta, lo que borra completamente su contenido. Para «grabar» un nuevo programa se coloca la EPROM borrada en un programador especial (figura de abajo). El programador permite aplicar una serie de pulsos sincronizados a las patas de la EPROM, al tiempo que le suministra los datos que se desea grabar, de octeto en octeto. El de la figura es más complejo.

Taco conductor. Este taco de espuma sirve para evitar que la electricidad estática pueda dañar el *chip*.

Conectores de red y TV. Visualizan el contenido de la EPROM en pantalla.

Conexión al ordenador. Permite conectar el programador a un ordenador para comprobar el programa antes de grabarlo en el *chip*.

Interruptores de selección. Permiten seleccionar distintas operaciones en el programador.

Zócalo. Aquí se colocan las EPROM borradas que se van a reprogramar.

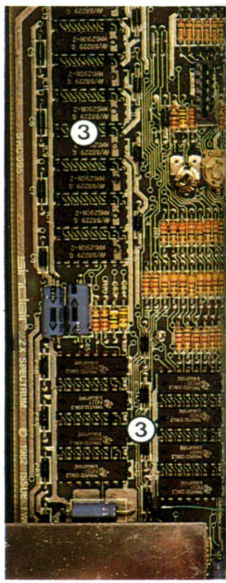
Teclado. Permite teclear un programa en hexadecimal.

Cuando la CPU quiere «escribir» un dato en una posición de memoria, se da un proceso análogo: pone la dirección de memoria en el *bus* de direcciones y el dato en el de datos. La dirección activa las puertas lógicas adecuadas y el dato acaba depositado en la posición de memoria encartada, reemplazando lo que antes pudiera haber en ella.

La RAM y la ROM

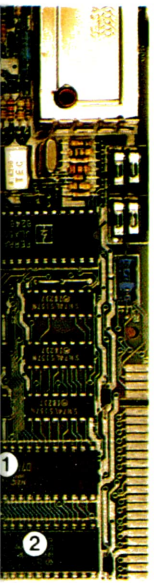
Como la CPU puede leer o escribir directamente en cualquier posición de memoria, a la memoria de este tipo se le llama «de acceso aleatorio» o RAM *(Random Access Memory)*. Sin embargo la utilización que la CPU hace de la RAM no tiene nada de aleatorio; el término sirve tan sólo para distinguirla de otros tipos más antiguos de memoria a la que se accedía en serie. La cantidad de RAM que una CPU puede «direccionar» tiene un límite, impuesto por el número de hilos de que consta el *bus* de direcciones. En la mayor parte de las CPU de 8 bits este número es 16, con lo que el máximo de posiciones de memoria «direccionables» es de 2^{16} (porque la dirección enviada es un número binario de 16 bits), o sea 65.536. Esto supone una capacidad de memoria de 64 koctetos, puesto que en cada posición hay un octeto. Los nuevos procesadores de 16 bits pueden «direccionar» memorias mucho mayores porque de la CPU salen más líneas de dirección.

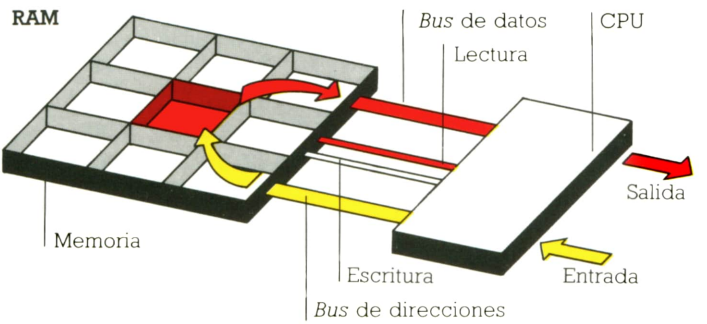
Los *chips* que componen la RAM tienen una desventaja: en cuanto apagamos el ordenador se pierde su contenido. Convendría pues contar con algún medio de dar instrucciones a la CPU en el momento de encender, aunque no sea más que para indicarle cómo cargar en la RAM desde un disco determinado *software*. Para esto suele usarse la memoria llamada «de sólo lectura» o ROM *(Read Only Memory)*. Podríamos considerar la ROM como una celdilla especial cubierta por un cristal. Podemos ver su contenido, introducido en el momento de fabricar las celdillas, pero no modificarlo. La lectura se realiza como en la RAM.

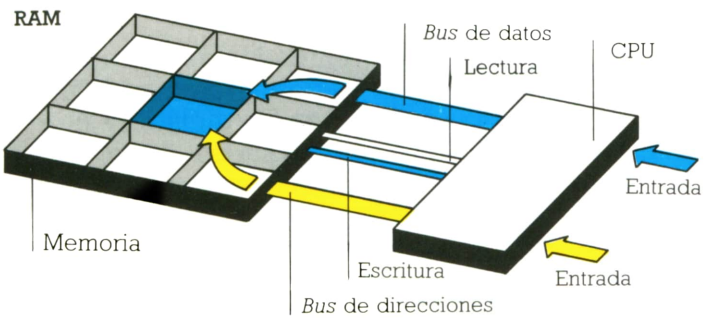


Funcionamiento de la memoria. Se muestra (derecha) la interacción entre la CPU y la RAM y ROM. La CPU lee los datos de una posición de RAM (arriba), enviando previamente su dirección al *bus* de direcciones para que de la zona de RAM activen al recibirla la celda de RAM que la CPU quiere leer. La CPU envía también una señal de control que indica a la celda de RAM que

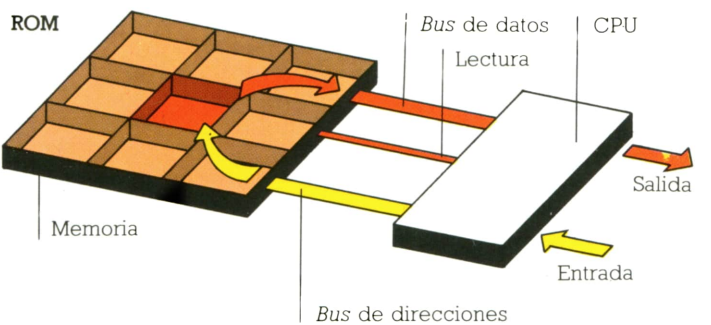
En el ordenador de abajo se distingue la CPU (1), que realiza todas las funciones de cálculo. Esta máquina lleva el sistema operativo y el intérprete de Basic en un único *chip* de ROM (2). Al encender la máquina, la CPU toma automáticamente instrucciones de la ROM. Los programas residen en RAM (3) y, cuando se da la orden RUN, el intérprete traduce el programa a código máquina.







se trata de leer, y no de escribir, datos en memoria. La operación de escritura (centro) es parecida, salvo en que la CPU envía una señal de control para escribir. En la ilustración de abajo se muestra la lectura de una celda de ROM. Como de una ROM se puede leer, pero no cambiar su contenido, no se precisa una señal de control que indique escritura.



ENTRADA Y SALIDA

Todo ordenador necesita comunicarse con el mundo exterior. En su forma más sencilla, bastaría con un interruptor y una luz: se abre el interruptor para poner en marcha el ordenador, y éste enciende la luz para indicarnos que ha terminado su trabajo. No parece éste, obviamente, un modo de interacción muy completo.

Teclados y pantallas

El mínimo esencial de posibilidades de entrada y salida (E/S) de que debe disponer un sistema informático básico lo constituyen un teclado y una pantalla. El teclado más tosco, que se encuentra en los ordenadores domésticos más baratos, es el del tipo sensible al tacto. La mayor parte de los ordenadores de gestión llevan teclados análogos a los de las máquinas de escribir, algunos con teclas suplementarias, capaces de generar órdenes complicadas con sólo pulsarlas una vez.

El monitor es parecido a un televisor y está conectado a unos circuitos especiales que le permiten presentar caracteres en función de sus códigos ASCII. Los ordenadores de gestión suelen utilizar un monitor especial, mientras que los domésticos suelen conformarse con un televisor.

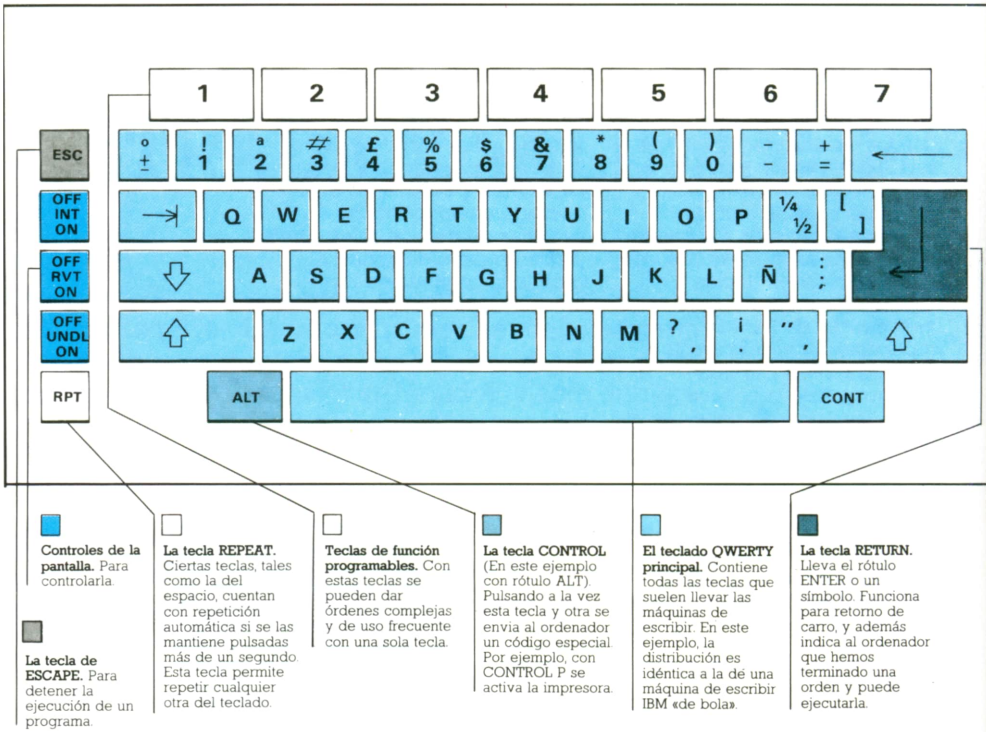
Puede considerarse la pantalla como una ventana que nos permite contemplar cierta zona de RAM dedicada a esta función. Unas veces, esta zona ocupa parte de la RAM principal de la CPU, otras es un bloque de memoria independiente al que accede la CPU con ayuda de ciertos circuitos suplementarios.

Teclados de Sinclair. En 1980, Clive Sinclair presentó en el Reino Unido el ordenador más barato del mundo, el ZX80. Le seguiría un año más tarde el ZX81, comercializado en Estados Unidos con el nombre de TS1000, versión muy perfeccionada. Ambas máquinas poseían teclados sensibles al tacto, en los que en lugar de teclas móviles hay una membrana con las teclas marcadas encima. Aunque menos caros de fabricar e impenetrables al polvo, estos teclados son de uso muy engorroso. Este problema queda atenuado en parte por el uso de teclas que generan una instrucción completa del Basic. Por desgracia, hay más palabras en Basic que teclas en el teclado, por lo que a veces, para generar palabras de uso menos frecuente, es preciso pulsar una secuencia de teclas.

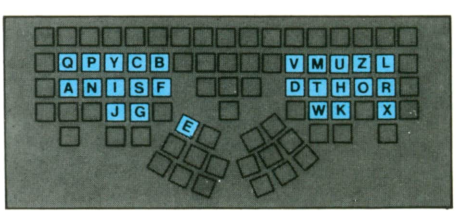
El ZX Spectrum, de Sinclair, sigue el mismo sistema, pero el teclado cuenta con teclas móviles, moldeadas en una sustancia blanda como de goma. Como las teclas del Spectrum son casi del mismo tamaño que las de una máquina de escribir pequeña, es posible utilizarlo en aplicaciones sencillas de gestión, tales como procesamiento de textos. El problema continúa en las limitaciones de su pantalla.

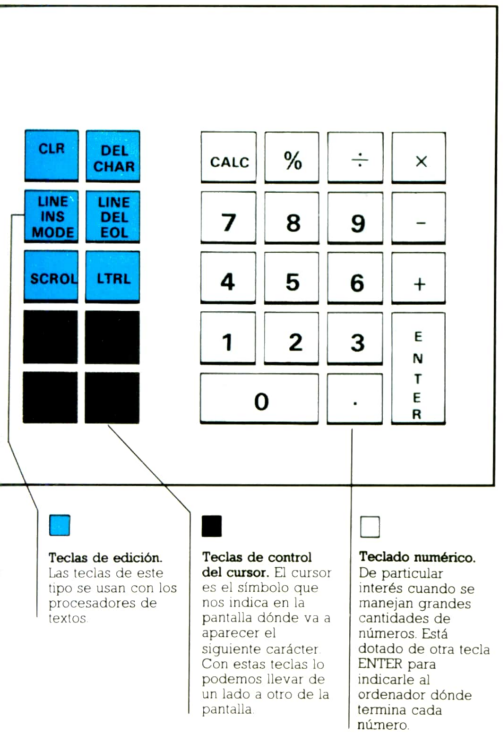
ZX81

ZX Spectrum



Teclado Maltron. El teclado QWERTY fue creado con la idea de dificultar a los mecanógrafos el desarrollo de altas velocidades, pues las primeras máquinas de escribir eran lentas para una persona experta. Esta distribución, sin embargo, es la que ha prevalecido, aunque se haya intentado sustituirla por otros diseños, como el Maltron. Teóricamente hace posibles velocidades mayores y resulta más fácil de aprender que el QWERTY.





Teclas de edición. Las teclas de este tipo se usan con los procesadores de textos.

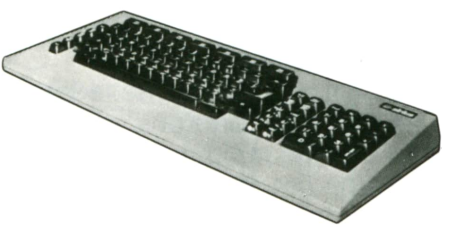
Teclas de control del cursor. El cursor es el símbolo que nos indica en la pantalla dónde va a aparecer el siguiente carácter. Con estas teclas lo podemos llevar de un lado a otro de la pantalla.

Teclado numérico. De particular interés cuando se manejan grandes cantidades de números. Está dotado de otra tecla ENTER para indicarle al ordenador dónde termina cada número.

TECLADOS DE ORDENADOR

En el teclado el usuario introduce sus órdenes o sus respuestas a las peticiones del ordenador. Algunos micros domésticos llevan teclados sensibles al tacto, que resultan útiles para juegos o para teclear programas cortos, pero aplicarlos a tareas de gestión resultaría muy tedioso. Las máquinas domésticas más completas, y la totalidad de los ordenadores de gestión, utilizan el tradicional teclado de máquina de escribir, a veces con teclas suplementarias que facilitan ciertas operaciones con el ordenador y realizan otras funciones que no existen en las máquinas de escribir. Muchos micros cuentan asimismo con teclas especiales llamadas «teclas de función programables», gracias a las cuales es posible dar a la máquina órdenes complejas pulsando una sola tecla. Son especialmente útiles para órdenes usadas con mucha frecuencia.

Teclado de gestión. En la foto, un teclado típico de gran calidad. En el diagrama de arriba se muestran las clases de teclas y sus funciones.



Mapas en memoria

Por regla general, la memoria se halla organizada de manera tal que a una posición específica de RAM le corresponde un lugar específico de la pantalla. Esta es la RAM de pantalla, que permite a la CPU poner caracteres en el lugar de la pantalla que desee. Este proceso se denomina «con mapa de la pantalla en memoria».

Los circuitos de vídeo del ordenador presentan en pantalla el contenido de la RAM, interpretando cada código ASCII como una dirección en la cual examinar una zona de ROM que contiene el conjunto de puntos que configura el carácter, y convirtiendo este conjunto en una señal de vídeo. Así pues en algún lugar de la ROM están los conjuntos de puntos que configuran los distintos códigos ASCII.

Un mapa en memoria más avanzado lo constituye la puesta en correspondencia de los bits de la RAM y los puntos de la pantalla; un 1 binario en un bit provoca la iluminación del punto correspondiente, un 0 su apagado. Es lo que se denomina «mapa de bits» y dota a la CPU de un mayor control sobre la pantalla, posibilitando el uso de los gráficos de alta resolución. Esto exige un *software* más complejo, y algunas máquinas precisan reservar el triple de la cantidad de memoria normal para almacenar los componentes rojo, verde y azul de una imagen en color.

Interfaces

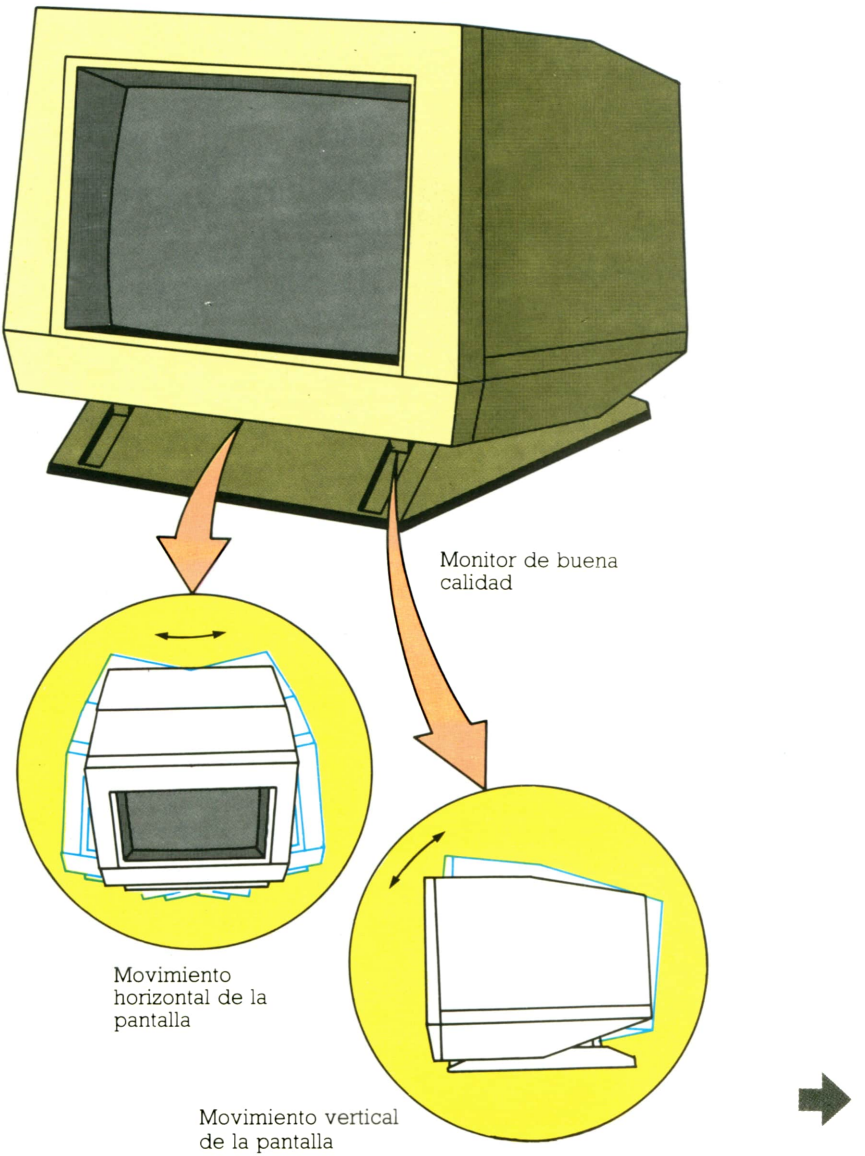
Normalmente los ordenadores han de comunicarse con otros equipos, aparte de pantallas y teclados. Como mínimo, la mayoría de las máquinas de gestión llevan una impresora, y hay dos maneras de enviar información desde el ordenador a la impresora (o a otros dispositivos): en serie y en paralelo. Ya hemos visto que los ordenadores almacenan datos en código binario (véase Módulo 13) y que la mayor parte de los microordenadores se organizan en torno a unos bloques de ocho dígitos binarios llamados octetos. Existe además un

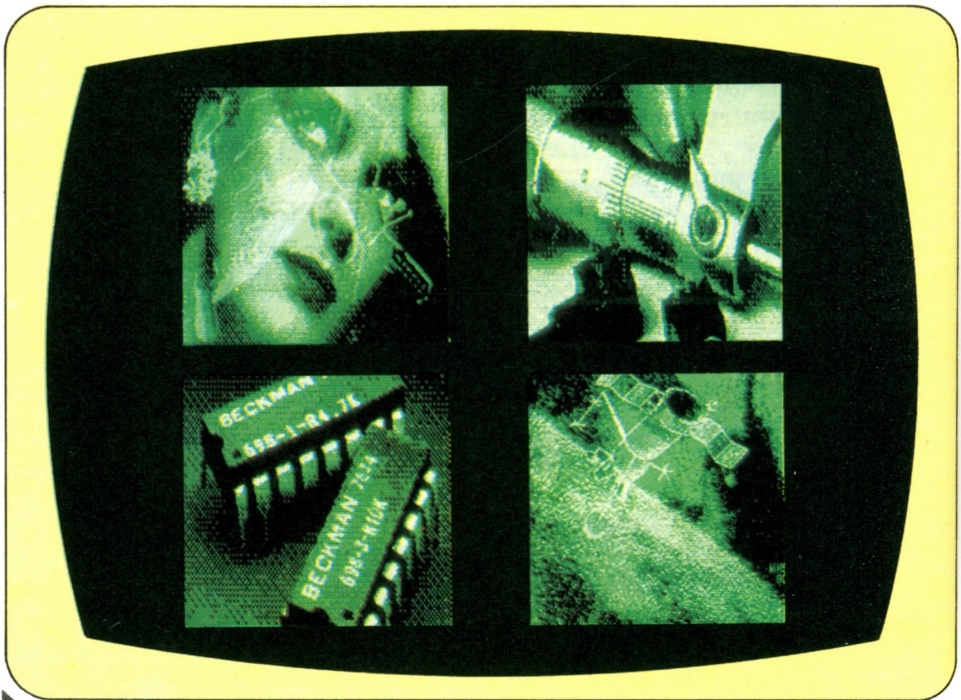
MONITORES O PANTALLAS

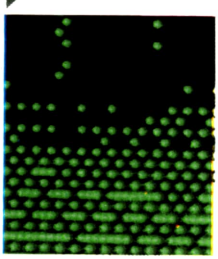
El medio más usual de comunicación del ordenador al usuario es el monitor o pantalla. Con aspecto de televisor, es capaz como mínimo de presentar letras y números. La mayor parte de los

Un monitor típico. En las aplicaciones de gestión es frecuente que una persona haya de permanecer delante de la pantalla varias horas diarias, y por ello cuenta mucho la calidad del monitor: pantalla antirreflectante (elimina reflejos fatigosos), y posibilidad de buscar el ángulo de contemplación más cómodo para el operador moviendo la pantalla en vertical o en horizontal.

micros domésticos y de gestión pueden mostrar asimismo símbolos gráficos. Son cada vez más los micros que cuentan con pantallas de color. El color atractivo para juegos, en la empresa no es un lujo: es un nuevo canal de información, pues los gráficos en color son más eficaces.

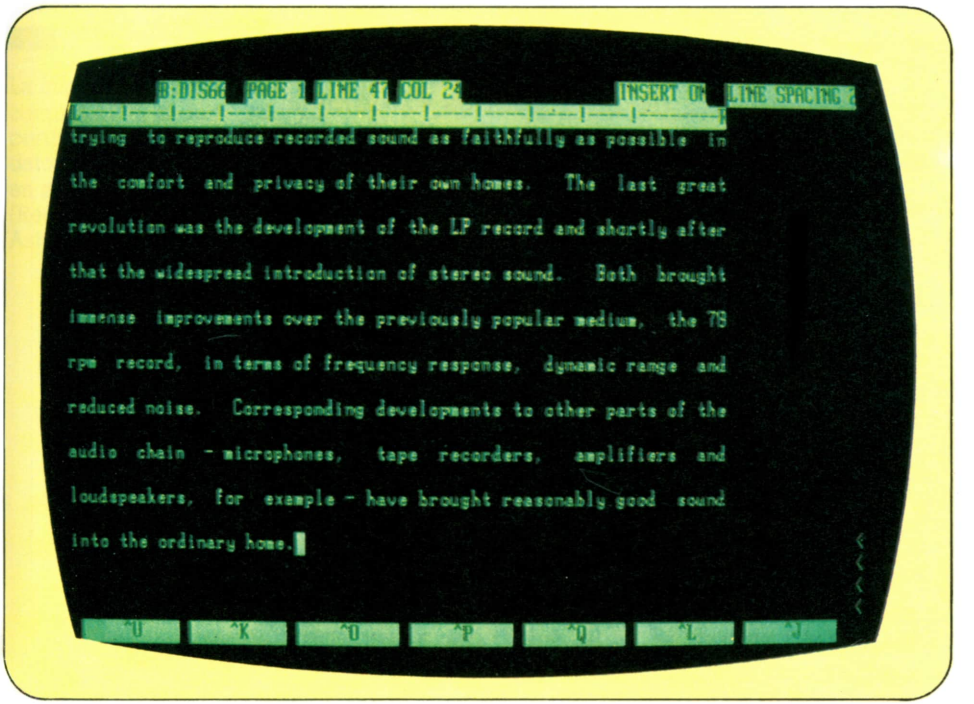




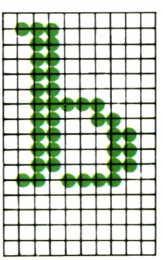


La imagen gráfica de la pantalla está formada por pequeños puntos: cuantos más, más detallada la imagen. Las cuatro imágenes de arriba se han obtenido de una pantalla con una resolución de 800 puntos horizontales por 400 verticales. Cada punto de la pantalla se corresponde con un bit de la memoria del ordenador: un 1 en la memoria hace que se ilumine el punto correspondiente en la pantalla y un 0 que se apague.

código normalizado para representar caracteres en binario, el ASCII. Aunque los números pueden almacenarse en el ordenador con un formato diferente, también hay que convertirlos en una cadena de códigos ASCII para poder presentarlos en pantalla o enviarlos a una impresora. Parece pues necesario y conveniente enviar información



Textos en pantalla. Los caracteres procedentes de un procesador de textos aparecen en pantalla por un procedimiento análogo, pero es preciso «traducir» primero los códigos ASCII de los caracteres a la configuración de puntos adecuada. Algunos ordenadores lo hacen dentro de los circuitos de vídeo, por *hardware*, y otros mediante técnicas de *software*. Con éstas resulta más fácil modificar los caracteres.



a una impresora de octeto en octeto y en código ASCII.

El ordenador se comunica a través de los puertos de entrada y salida, y en algunas máquinas estos puertos son para la CPU posiciones de memoria. Este sistema «de mapa de E/S en memoria» exige el sacrificio de ciertas zonas de memoria

para evitar el maremágnum electrónico que puede organizarse si una posición de memoria y un puerto de E/S tienen la misma dirección. En otras máquinas, la CPU emite instrucciones especiales para trasladar datos de unos puertos externos al mapa de memoria y viceversa.

Una vez llegados los datos al puerto, hay que transmitirlos al dispositivo externo ateniéndose a alguna norma, para que los fabricantes de periféricos puedan construir equipos compatibles con cualquier máquina. Uno de los métodos más populares es la *«interface en serie RS232»*.

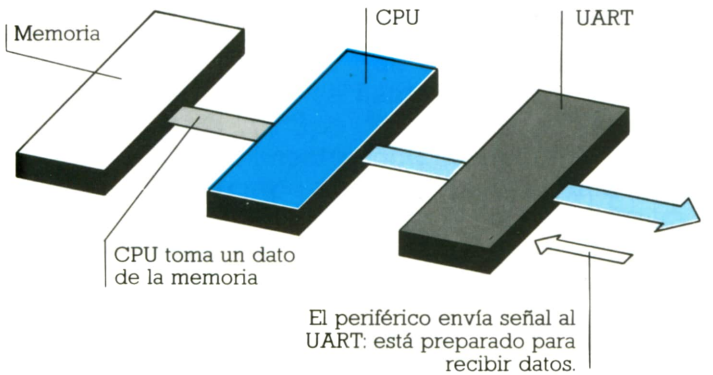
Interfaces en serie

Una *interface* en serie toma un octeto del ordenador y lo transmite, bit a bit, por medio de un *chip* llamado Transmisor-Receptor Asíncrono Universal o UART *(Universal Asynchronous Receiver/Transmitter)*. El UART ha de hacer varias cosas: añadir algunos bits complementarios que indiquen al dispositivo receptor que se va a transmitir un dato, enviar el dato e indicar al receptor que el dato ya está enviado.

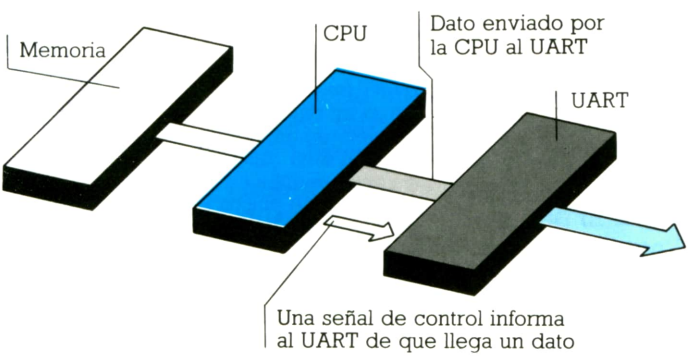
En primer lugar, transmisor y receptor han de ponerse de acuerdo en cuanto a la velocidad a que van a transmitirse los datos. Es lo que se llama velocidad en «baudios». Baudio significa «unidades de información por segundo» y, a grosso modo, el número de baudios dividido por diez equivale al número de caracteres por segundo. Después es preciso ponerse de acuerdo en cuanto al «protocolo» para que el ordenador sepa cuándo tiene que enviar los datos y el periférico sepa cuándo le van a enviar algo. El método más sencillo es que el UART ponga al principio un «bit de comienzo» que indique al receptor que los siguientes ocho bits forman un carácter, y añada un «bit de parada» al final, para dar a entender que el dato ha concluido. Para que tenga lugar una comunicación correcta es preciso que ambos dispositivos acuerden el número de bits suplementarios que van a usar. Por último, ha de haber una

INTERFACE EN SERIE

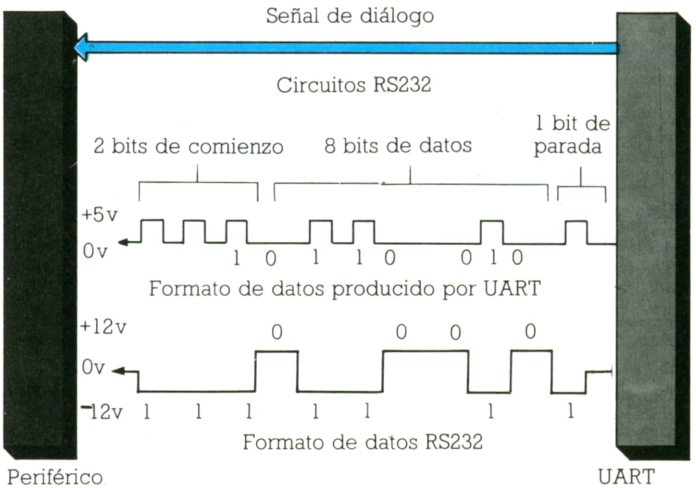
La *interface* en serie permite comunicar el ordenador y un periférico, transmitiendo y recibiendo datos bit a bit. El núcleo de la *interface* en serie es el UART (Receptor-Transmisor Universal Asíncrono). Mediante este dispositivo se transforman a modo serial los datos en paralelo usados en el ordenador. Se trata de un dispositivo bidireccional, porque también recibe datos en serie del periférico y los convierte al formato paralelo usado en los circuitos internos del ordenador.

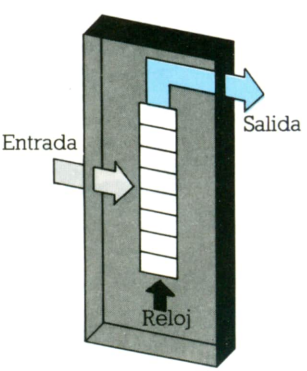


La secuencia de salida. Para que el ordenador saque los datos por un puerto en serie (abajo), la unidad central de proceso los va leyendo de la memoria octeto a octeto y los envía, a través del UART, al *bus* de datos. Unas señales de control le dicen a la unidad central de proceso que el UART está listo para recibir datos, y también la CPU ha de indicar al UART que va a transmitir un dato.



La *interface* RS232. Se trata de una *interface* estándar para la comunicación en serie entre ordenadores y periféricos. Los datos en serie suelen salir del UART en forma de un pulso de +5 voltios para el 1 binario y 0 voltios para el 0 binario. Antes de enviarlos a un periférico, los circuitos de la *interface* RS232 los convierten a -12 voltios para el 1 binario o +12 voltios para el 0.





El UART. En el UART se carga un dato en paralelo en un registro de desplazamiento. Una serie de pulsos procedentes de un reloj electrónico hacen que el contenido del registro se desplace un bit por cada pulso.

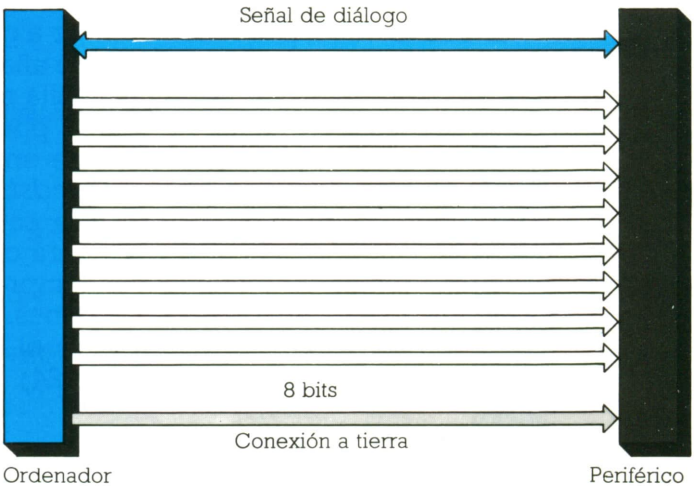
norma eléctrica común a los dos dispositivos partícipes en la comunicación. Ordenador y periférico pueden por tanto estar conectados por sólo tres hilos: la tierra común, el que porta los datos que vienen del ordenador y el que porta los datos que llegan del periférico. Con la norma RS232 el sistema de tres hilos funciona bien hasta una cierta velocidad, pero para trabajar a grandes velocidades suele ser imprescindible añadir más hilos para que, si el periférico necesita más tiempo para procesar los datos recibidos, pueda decirle al ordenador que espere antes de enviarle más. Es lo que se llama «diálogo» *(handshaking)*. Tal vez este tipo de *interface* en serie parezca innecesariamente complicado, pero para cosas sencillas (como mandar un texto a una impresora) abarata mucho el coste de las conexiones, sobre todo para largas distancias, empleando el dispositivo llamado «módem» (véase Módulo 24).

Interfaces en paralelo

También es posible enviar los datos de ocho en ocho bits, empleando un hilo para cada bit. Es lo que se llama «en paralelo», siendo frecuente su uso en el manejo de impresoras con una *interface* estándar llamada «tipo Centronics». Para trabajar con este sistema se precisan diez hilos: uno para la tierra, ocho para los datos y otro para el diálogo. El envío de datos por una *interface* en paralelo es obviamente mucho más rápido en cuanto a la transmisión real de datos, pero sigue siendo necesario que la impresora indique al ordenador que espere mientras termina de imprimir los caracteres. Muchas impresoras cuentan con un tampón *(buffer)*, una pequeña cantidad de memoria en la que puede almacenarse una línea entera. La impresora aguarda hasta que le haya llegado toda la línea, colocando los caracteres en el tampón, y entonces le dice al ordenador que espere a que imprima esa línea. Hay impresoras que tienen tampones mucho mayores, y se venden sueltos algunos que alcanzan hasta los 64 koctetos.

INTERFACE EN PARALELO

Enviar datos en paralelo necesita diez hilos: uno para cada bit, y dos más para la tierra y el diálogo (véase abajo). Uno permite al periférico indicar que está listo para recibir datos y el otro lleva la señal del ordenador de que va a transmitir un octeto al periférico.



Otra norma para *interfaces* en paralelo es la IEEE-488 o *Hewlett-Packard General Purpose Interface Bus.* Algunas máquinas la usan para comunicarse con periféricos, pero su aplicación más habitual se da en los casos de equipos de laboratorio conectados a un ordenador para control o adquisición de datos. Es más complicada (y flexible) que la *interface* Centronics, pues permite la conexión de varios equipos al mismo puerto de E/S. Cada dispositivo tiene su dirección, y el ordenador tiene que enviar primero la dirección del dispositivo al que piensa mandar los datos (o del que piensa recibirlos) para activarlo y, al mismo tiempo, desactivar los demás que halla conectados al mismo puerto. Con esto el número de periféricos que el ordenador puede manejar aumenta considerablemente, a expensas de más electrónica y de un *software* más complejo.

CASSETTES Y CARTUCHOS

Se han ideado varios métodos para almacenar programas y datos de forma que el ordenador pueda leerlos directamente. Uno de estos métodos es el disco (véase Módulo 22), cuyo uso va extendiéndose paulatinamente entre los usuarios de ordenadores domésticos, pero estas máquinas, en su inmensa mayoría, también pueden guardar programas y datos en *cassettes*

Almacenamiento de datos en *cassettes*

Si el diseñador del *hardware* ha hecho un buen trabajo podremos guardar nuestros programas y datos casi con cualquier magnetófono que tengamos. Se pueden usar *cassettes* de audio corrientes, pero es más seguro utilizar *cassettes* digitales especialmente preparadas, que se fabrican con unos controles de calidad mucho más exigentes.

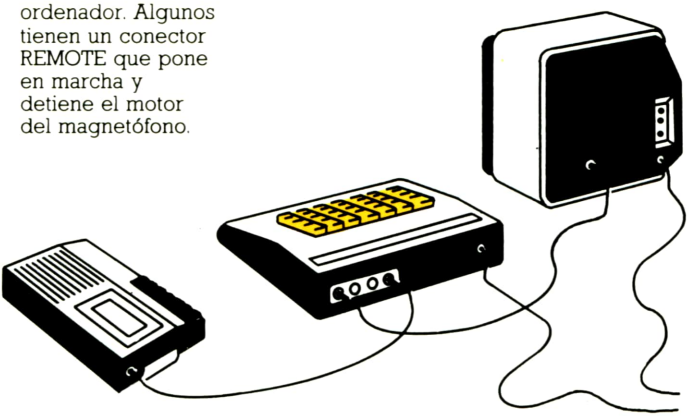
Los magnetófonos presentan varias desventajas que para aplicaciones de gestión los hacen casi inútiles. Primero, son lentísimos: las velocidades de los magnetófonos oscilan entre los 300 y los 2.400 baudios (véase Módulo 20), pero ni siquiera los más rápidos dejan de ser más lentos que los discos.

Sin embargo, el mayor inconveniente del magnetófono estriba en que sólo permite el acceso secuencial a los datos. Así por ejemplo, si hemos escrito un fichero en una *cassette* y queremos leer, por ejemplo, el elemento número 50, tenemos que esperar a que la *cassette* recorra a su lentísimo ritmo los 49 anteriores. Si luego queremos hacer referencia al elemento número 20, habrá que rebo-

MAGNETOFONOS

Los datos se envían al magnetófono pasando por un UART (véase Módulo 20) y luego por unos circuitos que transforman los ceros y unos en tonos de audio. Otros circuitos de la *interface* realizan lo inverso, recogiendo los tonos que el magnetófono lee y restaurando los ceros y unos. El UART los reagrupa en octetos y pasan luego al *chip* de la CPU. Conviene cerciorarse de que nuestro *cassette* cuenta con conectores para entrada de micrófono y salida a auricular, porque son necesarios para conectarlo a un ordenador.

Conexión. El conector del micrófono del magnetófono se conecta al MIC o TAPE OUT del micro, y el conector del auricular al EAR o TAPE IN del ordenador. Algunos tienen un conector REMOTE que pone en marcha y detiene el motor del magnetófono.



binar la cinta, volver al principio, y pasar otra vez por los 19 primeros.

Se ha ensayado la fabricación de sistemas de *cassettes* que resuelven estos problemas. Uno de ellos, llamado *stringy floppy* (disco flexible en cinta), utiliza un bucle de cinta continua que le permite acceder individualmente a cada elemento. Con este método, después de leer el elemento 50, el sistema espera automáticamente a que vuelva a pasar el número 20, ahorrando así el tiempo necesario para rebobinar la cinta. Funcionaba muy bien, era mucho más rápido que los sistemas corrientes de *cassette* y tenía fama de fiable, pero nunca llegó a gozar de popularidad.

Se han ideado también otros sistemas en los que todo el mecanismo del aparato de *cassette,* incluyendo el avance y rebobinado rápidos, está bajo control del ordenador para que éste sepa exactamente en qué lugar de la cinta se halla en un momento determinado.

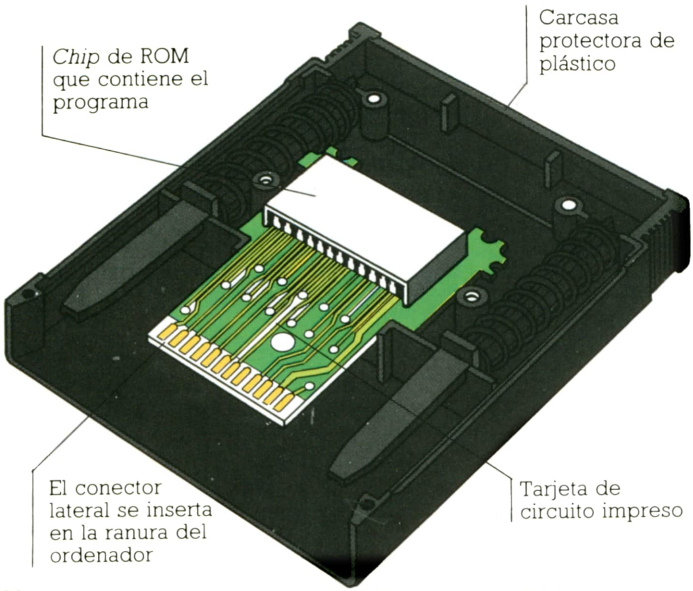
A los creadores de *software,* además, las *cassettes* les plantean un problema especial: el robo de *software,* llamado a veces «piratería». Copiar un programa comercializado de una *cassette* a otra suele resultar muy fácil, y aunque a veces se haga sin intención expresa de robo (lo más corriente es el intercambio inocente entre amigos), al cabo supone privar de su dinero a las personas que han invertido su esfuerzo en desarrollarlo.

Cartuchos de ROM

Los dos problemas de las *cassettes* se resuelven fijando el *software* permanentemente en uno o más *chips* de ROM. Las ROM se montan en una pequeña tarjeta de circuito impreso, enlazando todas sus conexiones a un conector especial situado en uno de los lados de la tarjeta. Luego se introduce en una carcasa de plástico y lo que resulta es lo que suele llamarse un cartucho de ROM.

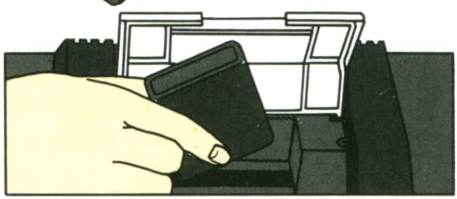
Muchos ordenadores domésticos llevan una ranura en la parte posterior, o a un lado, donde pueden enchufarse estos cartuchos de ROM. Cada

CARTUCHOS



Las *cassettes* tienen dos inconvenientes: por un lado son lentas y por otro resulta fácil copiar una cinta, con lo que el robo de *software* se convierte en un verdadero problema. Para resolverlo, muchos micros llevan una ranura en la que se pueden conectar cartuchos de *software*. Los cartuchos son pequeños contenedores de plástico que albergan una tarjeta de circuito impreso en la que van montados uno o varios *chips* de ROM que contienen el programa.

Programa que suele ser un juego o una aplicación más «seria», por ejemplo un procesador de textos. Una vez instalado el cartucho, el programa se activa al encender el ordenador o tecleando una orden sencilla.



Inserción del cartucho. En la ilustración aparece la posición normal de un cartucho en un microordenador.

marca de máquina acepta sólo los cartuchos diseñados específicamente para ella. Para usar el cartucho se apaga la máquina, se introduce en la ranura, se vuelve a encender, y o la máquina empieza sin más a ejecutar el programa del cartucho o tenemos que activarlo nosotros tecleando una orden especial.

LOS DISCOS

Aunque ya hemos visto que las *cassettes* pueden servir para almacenar programas (véase Módulo 21), están sometidas a graves limitaciones, sobre todo en aplicaciones de gestión.

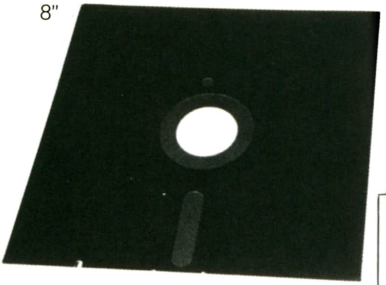
Los discos magnéticos resuelven los problemas de los sistemas de almacenamiento en cinta y son de empleo universal en los microordenadores de gestión, aparte de ser complemento opcional de un número cada vez mayor de máquinas domésticas. En principio, los discos son un cruce entre un magnetófono y un tocadiscos. El disco está revestido de un compuesto de óxido magnético similar al de las cintas de grabar. La unidad de disco (véanse las páginas 198-199)contiene un núcleo central que hace girar el disco, a 300 rpm en los *floppies*. Un brazo portador de una cabeza de lectura/escritura se desplaza radialmente sobre la superficie del disco bajo control del ordenador.

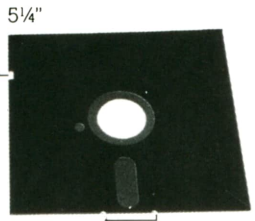
«Formateado» del disco

Para que el ordenador pueda encontrar fácilmente la información en el disco, su superficie se halla dividida en pistas concéntricas, cada una de las cuales está dividida a su vez en un número de sectores. No hay nada visible en la superficie del disco: lo crea el ordenador mismo en un proceso llamado «formateado» o inicialización. Es algo que es necesario hacer con cada disco antes de poder usarlo para almacenar y recuperar datos.

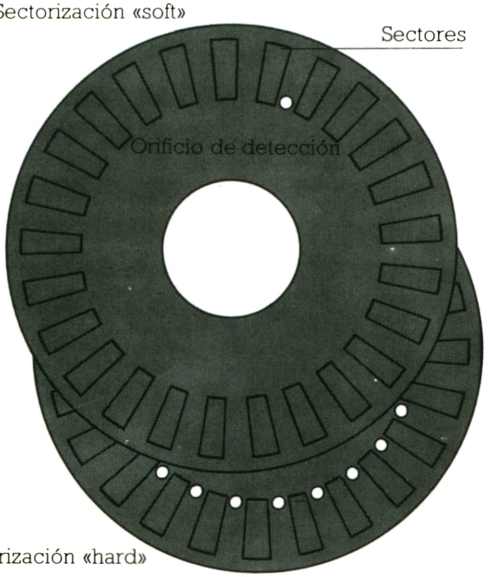
FLOPPIES

El *floppy* es el medio más utilizado para almacenar programas y datos en los micros de gestión, y se va popularizando entre los domésticos. Aunque los hay muy diversos, todos funcionan con arreglo al mismo principio: una mezcla de tocadiscos y magnetófono. El disco es de plástico revestido de óxido magnético y sellado luego en una funda protectora. La funda lleva una ranura por la que la cabeza de lectura/escritura de la unidad puede acceder al disco bajo el control del ordenador.





Sectorización «soft»

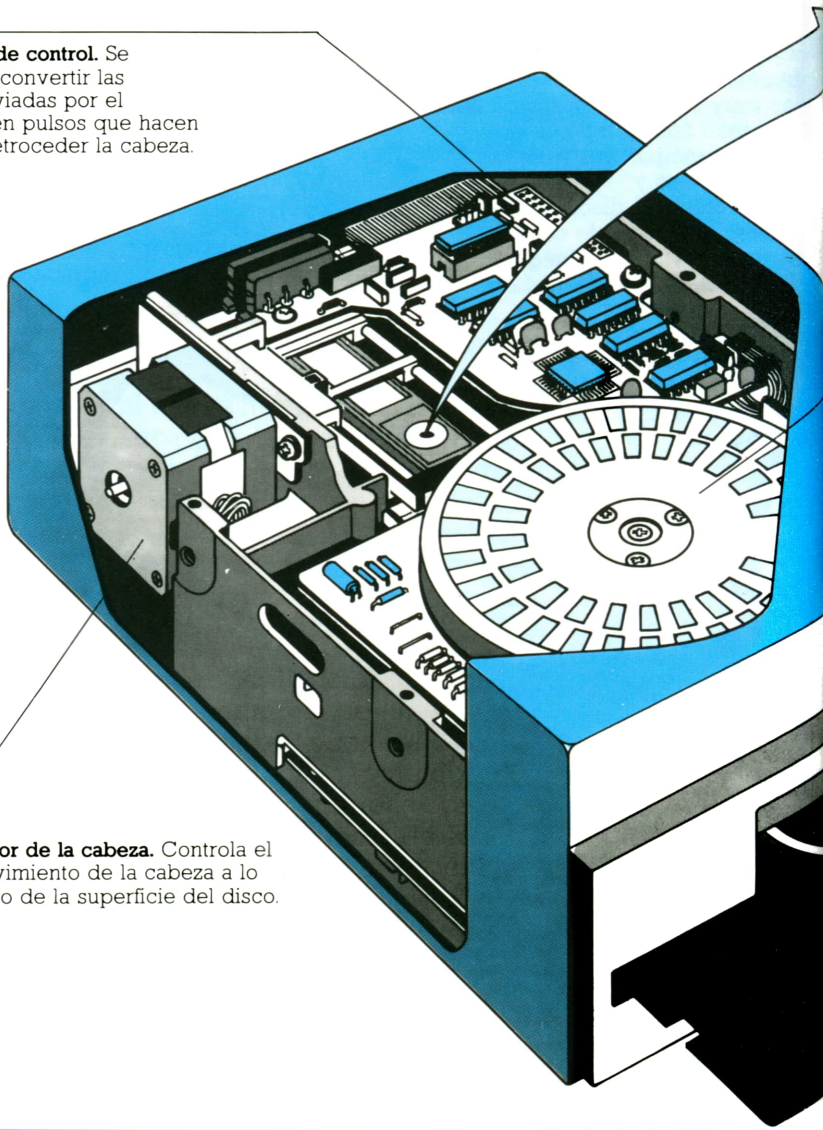


Sectorización «hard»

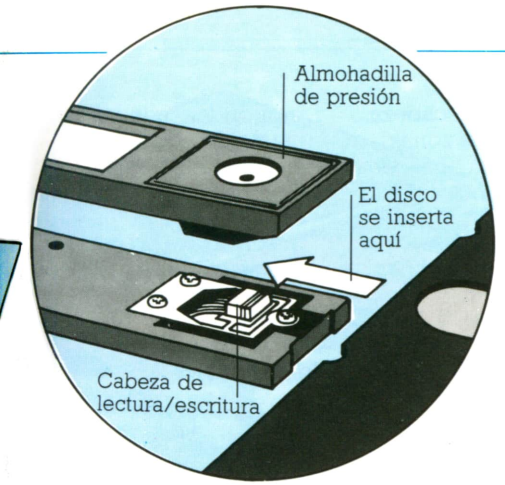
Tamaños y formatos. Hay *floppies* de 8, de 5¼ y de 3 ó 3½ pulgadas. La superficie del disco está dividida en pistas, y las pistas en sectores. Estas divisiones vienen marcadas por el ordenador durante un proceso de «formateado». Para que el ordenador pueda detectar el comienzo del primer sector de cada pista, hay un orificio cerca del centro del disco. Es lo que se llama sectorizado «soft», porque la grabación y detección de sectores se efectúa por *software*. En los discos sectorizados «hard» hay un orificio cerca del centro por cada sector.

El interior de la unidad de disco. Normalmente, la única parte visible de la unidad de disco es la ranura por la que se insertan los discos. Dentro, sin embargo, puede verse una combinación de complejos componentes electrónicos de control y mecánicos de precisión, imprescindibles para grabar datos en el *floppy* y leerlos de nuevo después. Al fondo de la unidad se halla la importantísima cabeza de lectura/escritura (véase detalle a la derecha).

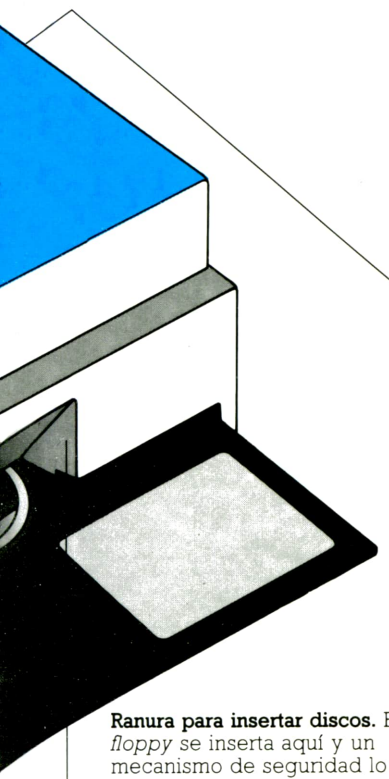
Electrónica de control. Se encarga de convertir las órdenes enviadas por el ordenador en pulsos que hacen avanzar o retroceder la cabeza.



Motor de la cabeza. Controla el movimiento de la cabeza a lo largo de la superficie del disco.



Cabeza de lectura/escritura. Avanza y retrocede sobre la superficie del disco en rotación. Se parece a la cabeza de un magnetófono en que convierte señales eléctricas en campos magnéticos. En la lectura se invierte el proceso. En una unidad de una sola cara, una almohadilla presiona el disco para que se mantenga en contacto con la cabeza. La sustituye, en una de doble cara, una segunda cabeza de lectura/escritura.

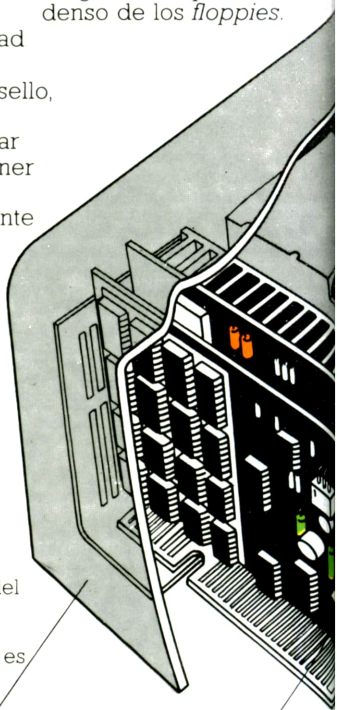


Motor del disco. Este motor hace girar el disco, normalmente a 300 revoluciones por minuto, aunque algunas máquinas cuentan con mecanismos de velocidad variable.

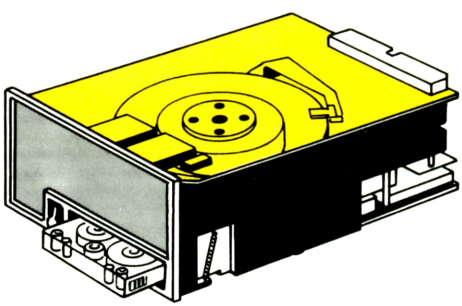
Ranura para insertar discos. El *floppy* se inserta aquí y un mecanismo de seguridad lo fija en su lugar.

DISCOS DUROS

La cantidad de información que se puede almacenar en un disco depende, entre otras cosas, de la velocidad de giro del disco y de la distancia entre su superficie y la cabeza de lectura/escritura. Los discos duros giran más deprisa que los flexibles, con lo que se crea una capa de aire en la que «vuela» la cabeza de lectura/escritura. Se reserva una zona especial del disco para que «aterrice» y «despegue» la cabeza al producirse el apagado y encendido. Si la cabeza toca una de las zonas utilizadas para almacenar datos, resultarán dañados tanto la cabeza como el disco. Para evitar que partículas suspendidas en el aire puedan adherirse al disco, la unidad se halla protegida por un contenedor sellado. Este sello, sin embargo, también impide que se pueda sacar un disco y meter otro. Poner dos discos para dedicar uno de ellos exclusivamente a copias de seguridad del otro sale caro.
En la actualidad, las unidades de disco duro proporcionan la mayor capacidad de almacenamiento, 5, 10, o incluso más megaoctetos comparado con un máximo de 1½ megaoctetos para el más denso de los *floppies*.

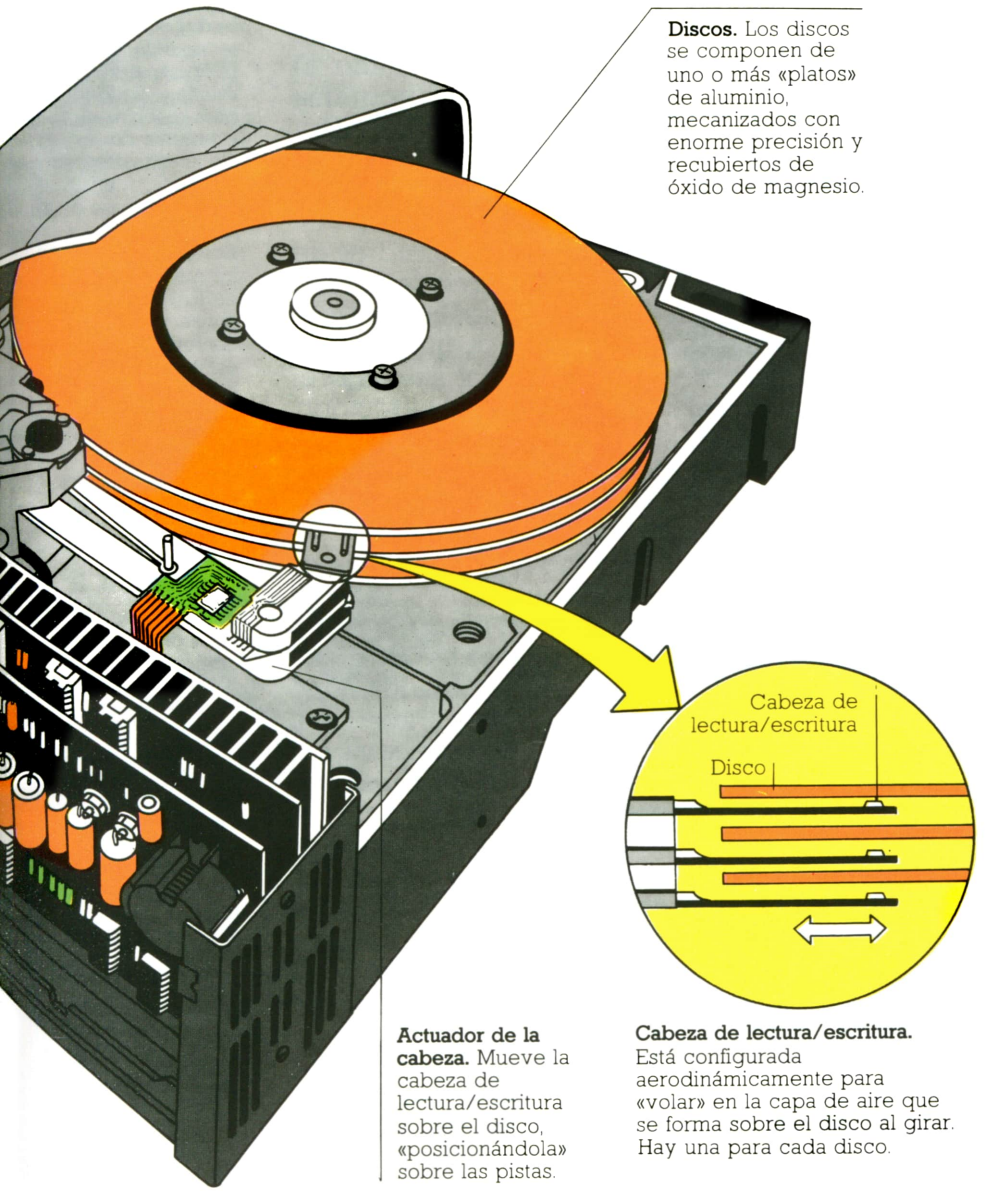


Copia de seguridad en cinta. Los discos duros tienen una gran capacidad de almacenamiento, hasta 20 o más megaoctetos. Sin embargo, tienen un inconveniente. Como están herméticamente cerrados, no es fácil copiar los datos a otro disco. Para evitar una posible pérdida de su contenido, podemos hacer copias usando multitud de *floppies*, o usar una unidad especial para copiar discos duros. Esta unidad se hace necesaria para ahorrar tiempo y además porque es probable que el tamaño del fichero que se quiere copiar exceda la capacidad de un solo *floppy*. Con esta unidad es posible copiar todo el contenido del disco duro.



Cámara. Una cámara herméticamente sellada protege el disco para impedir que penetren el polvo y la suciedad.

Tarjetas de circuitos. Se encargan de «traducir» las señales de control del ordenador en instrucciones para hacer que la cabeza avance y retroceda.



Discos. Los discos se componen de uno o más «platos» de aluminio, mecanizados con enorme precisión y recubiertos de óxido de magnesio.

Actuador de la cabeza. Mueve la cabeza de lectura/escritura sobre el disco, «posicionándola» sobre las pistas.

Cabeza de lectura/escritura. Está configurada aerodinámicamente para «volar» en la capa de aire que se forma sobre el disco al girar. Hay una para cada disco.

Directorio del disco

El ordenador guarda una relación de todo cuanto contiene el disco reservando un par de pistas para almacenar el «directorio» de dicho contenido. El directorio conserva el nombre de cada programa o fichero de datos y un «mapa» de las pistas y sectores que ocupa. En una máquina CP/M, el operador puede saber lo que hay en el disco tecleando DIR (abreviatura de DIRECTORY), pues en tal caso el ordenador mostrará todos los nombres de fichero y programa. Los ordenadores con otros sistemas operativos poseen órdenes análogas.

En una máquina CP/M, cuando se quiere usar un programa, primero hay que teclear su nombre. El ordenador buscará entonces en el directorio del disco un programa con ese nombre y, si lo encuentra, localizará, valiéndose del mapa, las pistas y sectores del disco donde realmente reside el programa. A medida que los va localizando los lee en memoria (véase Módulo 19) y, leído el último, el ordenador comienza a ejecutar el programa.

La escritura de información en el disco —es decir, el guardar datos o programas— se lleva a cabo mediante un proceso análogo. El ordenador, consultando la información del directorio, averigua qué pistas y sectores están libres, almacena en ellos los datos y elabora el mapa del nuevo fichero, añadiéndolo luego al directorio.

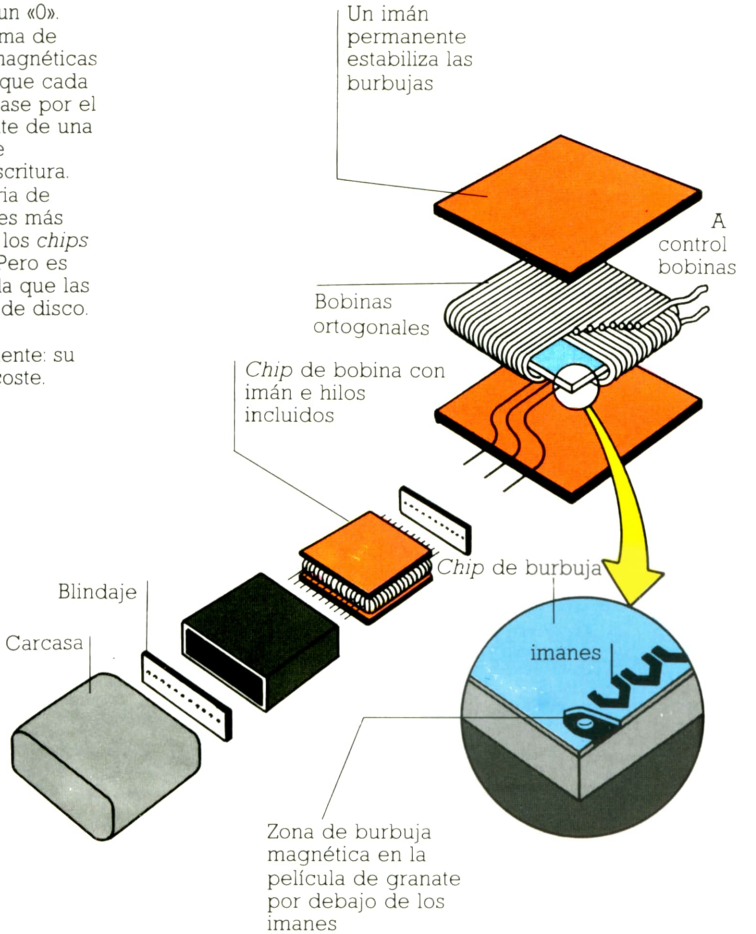
La técnica de almacenamiento de información en disco es bastante compleja, especialmente si recordamos que el ordenador, tras determinar a qué parte del disco quiere acceder, tiene que controlar el mecanismo de la unidad de disco. Lo consigue enviando señales que hacen que la cabeza de escritura/lectura avance o retroceda a lo largo del disco hasta llegar al lugar debido.

Tipos de discos

Suponiendo que tenemos cuidado en no dañar el *floppy* (disco flexible), es un método razonablemente fiable de almacenar datos. Los discos se desgastan, sin embargo, y por mucho cuidado que

Memoria de burbujas. Es una tecnología muy interesante y reciente. Se ideó para resolver el problema de almacenar datos en un medio robusto y de fácil acceso para el ordenador. Los discos son frágiles, y los *chips* de RAM sólo conservan su contenido mientras están enchufados. La memoria de burbujas, sin embargo, no pierde su contenido al cesar la alimentación y es muy sólida. Las «burbujas» son campos magnéticos diminutos en una capa de granate artificial emparedada entre imanes. Unas bobinas electromagnéticas pueden invertir las polaridades de las burbujas, con lo que una burbuja cuyo campo esté en una dirección tendrá un «1» y si está en dirección

contraria un «0». Otro sistema de bobinas magnéticas garantiza que cada burbuja pase por el equivalente de una cabeza de lectura/escritura. La memoria de burbujas es más lenta que los *chips* de RAM. Pero es más rápida que las unidades de disco. Principal inconveniente: su elevado coste.



llevemos, a veces se producen accidentes, por lo que siempre hay que conservar una copia de seguridad de la información en otro disco. La frecuencia con que debamos hacer esta copia dependerá de cuántos datos podamos permitirnos perder: si lo hacemos al acabar el día, sólo podremos perder la información nueva de un único día suponiendo que el disco falle inmediatamente an-

tes de ir a copiarlo. Los *floppies* tienen además otro inconveniente. La cantidad de información que pueden contener no es muy grande —oscila entre los 100 octetos y algo más de 1 megaocteto— y, en comparación con el ordenador, todavía son lentos (aunque viéndolos trabajar pueden parecernos bastante rápidos). Para solucionar estos problemas, algunas máquinas están hoy equipadas con «discos duros», como complementos opcionales o incluidos en la configuración básica.

El disco duro para microordenador más corriente es, en la actualidad, el disco de tipo «winchester». Estos discos duros constan de uno o más discos rígidos de aluminio dentro de un contenedor de plástico herméticamente sellado. En los discos duros se puede guardar la información más densamente en la superficie del disco. Esto se debe en parte a que el disco gira a mayor velocidad y en parte al diseño de la cabeza de lectura/escritura, que «vuela» sobre dicha superficie. Esta cabeza está diseñada aerodinámicamente para aprovechar la capa de aire móvil generada por el disco al girar, convirtiéndola en una especie de almohadilla que mantiene una distancia minúscula entre ella y la superficie del disco. Si la cabeza llegase a tocar la superficie abriría un surco, destruyendo toda la información almacenada en su trayectoria y dañando no sólo la cabeza y los datos, sino también el delicado revestimiento del disco.

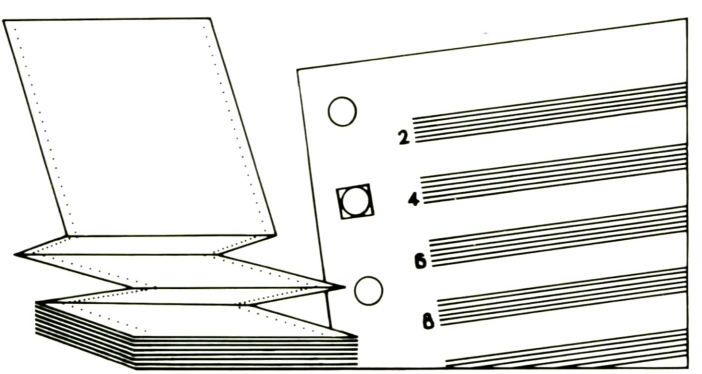
Una unidad de disco duro puede contener hasta 20 megaoctetos o más en un espacio no superior al de una unidad de *floppy,* y es muchas veces más rápida que ésta. Tiene, sin embargo, dos inconvenientes. En primer lugar, no es posible sacar un disco y reemplazarlo por otro, como ocurre en los *floppies.* En segundo lugar, aun siendo mucho más fiables que los flexibles, también pueden fallar. Por ello es preciso contar con algún medio de salvaguardar la información, ya sea con multitud de *floppies* (lo que crea un problema de almacenamiento), ya sea usando una unidad especial de copia en cinta (véase la página 200).

LAS IMPRESORAS

En un ordenador de gestión siempre hace falta escribir determinados resultados en papel: facturas, cartas, direcciones. Muchos aficionados compran también impresoras para sacar listados de los programas en que están trabajando, pues es más fácil detectar un error en un programa teniendo todas las sentencias en un papel que recorriéndolo poco a poco en la pantalla.

Tipos de impresoras

En informática se emplean hoy principalmente dos tipos de impresora: la matricial (véanse páginas 208-209) y la margarita (206-207). Preferir una u otra dependerá de la aplicación (las de margarita

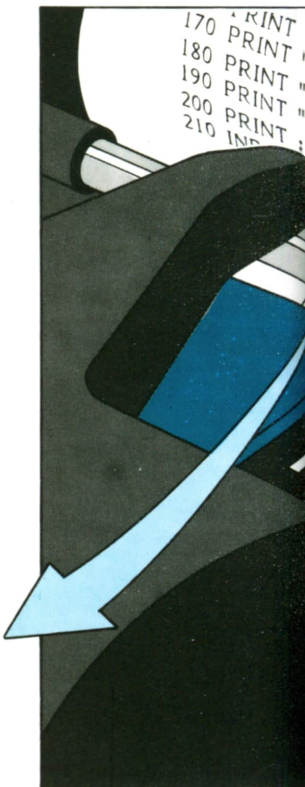


Papel para impresora. El tipo de papel a usar dependerá de la impresora. Las electrostáticas y térmicas exigen un papel especial, y no son adecuadas para aplicaciones de gestión. En las de margarita y en las matriciales hay más donde elegir. Está muy difundido el uso del papel continuo.

DE MARGARITA

Se utilizan para trabajos de calidad. Son caras y lentas en comparación con las matriciales. La margarita consiste en una rueda de plástico o metal con los caracteres moldeados en los extremos de los radios. Cuando el carácter que se quiere imprimir alcanza su posición, un martillo electromagnético lo presiona contra una cinta entintada.

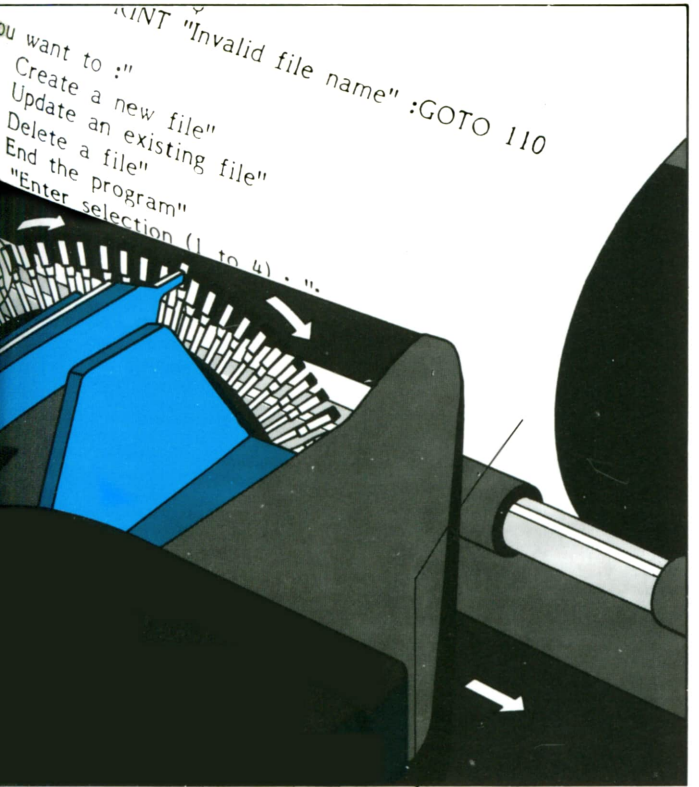
Tipos. Lleva los caracteres moldeados en los extremos de los radios de la rueda.



Cartucho de cinta. La cinta entintada viene en un cartucho fácil de cambiar.

producen unas letras de mayor calidad) y del dinero que se quiera gastar (las matriciales son mucho más baratas).

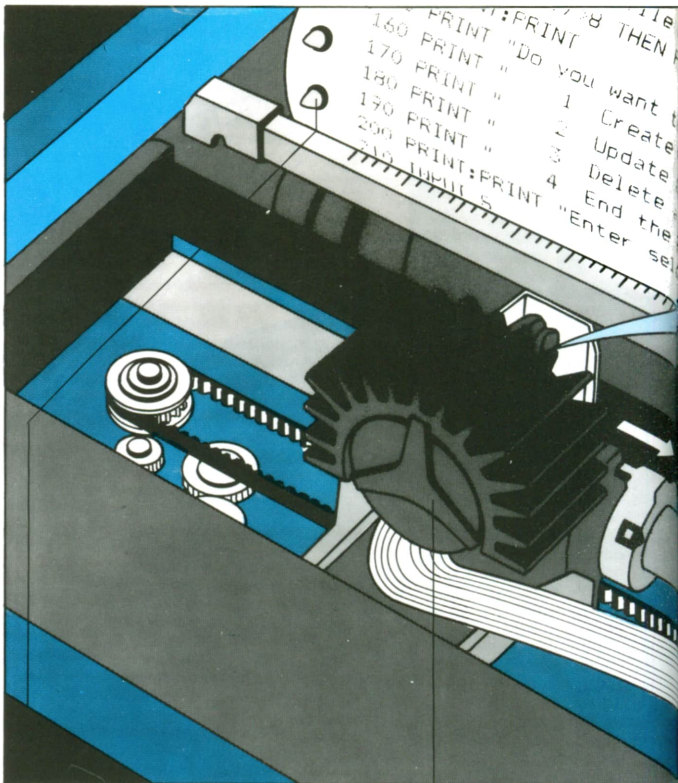
Como su nombre indica, la impresora matricial forma los caracteres sobre el papel a partir de una matriz de puntos. Lo más corriente es que una línea vertical de siete o nueve agujas (actuadas



Martillo. Gira la rueda hasta que el carácter elegido queda arriba, y, en ese momento, un martillo le presiona contra la cinta y el papel.

Arrastre del papel. Casi todas las impresoras de margarita usan arrastre por fricción.

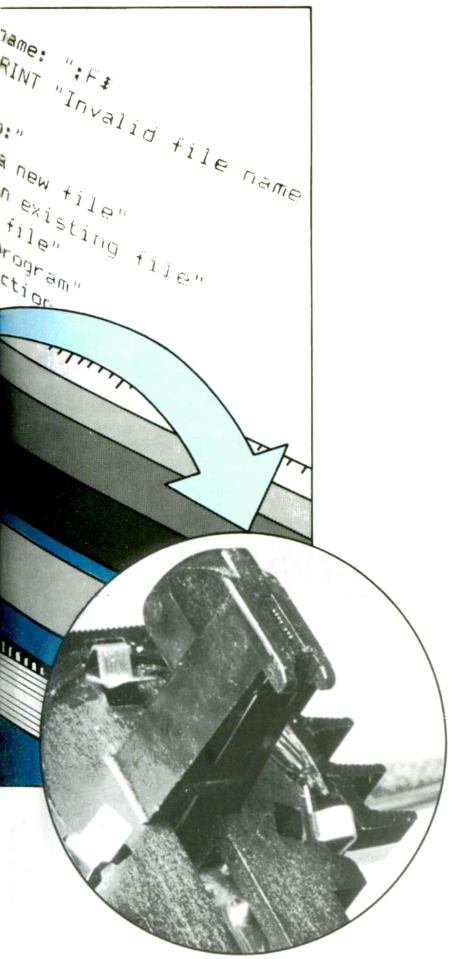
cada una de ellas por un solenoide) recorra el papel en una serie de pasos muy cortos. En cada uno de ellos se «adelantan» hacia el papel las agujas que se quiere que produzcan puntos. Delante de ellas hay una cinta entintada, lo mismo que en las máquinas de escribir, y los caracteres se forman punto a punto.



Arrastre del papel. La mayoría de las impresoras matriciales llevan un sistema de tracción que ensarta los agujeros del papel continuo.

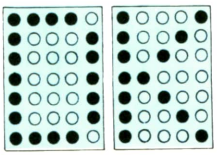
Cabeza de impresión. La cabeza lleva una columna de 7 ó 9 agujas; cada una de ellas tiene un electroimán que le hace golpear la cinta.

El problema principal de las impresoras matriciales es que imprimen caracteres con perfiles discontinuos, lo que no importa mucho en borradores o en contabilidad, pero no parece adecuado, por ejemplo, en una carta para un cliente. Los fabricantes de impresoras han invertido mucho tiempo y esfuerzo en mejorar la calidad de impre-



MATRICIALES

Los caracteres se configuran con una matriz de puntos. Se imprime columna a columna por medio de una cabeza de impresión móvil que tiene una línea vertical de agujas. A medida que va desplazándose la cabeza a lo largo del papel se adelantan las agujas necesarias para formar el carácter. Suelen ser más rápidas y baratas que las de margarita.
La calidad de impresión no es tan buena.



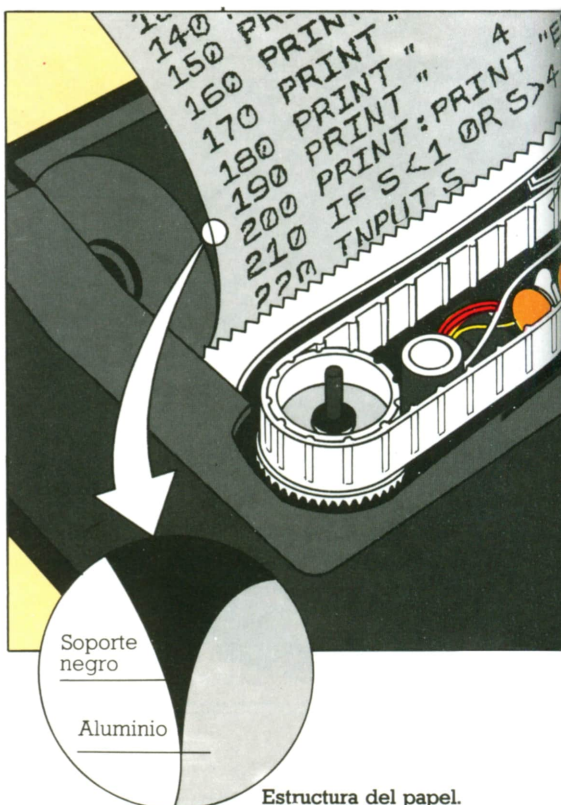
Detalle ampliado de la cabeza de impresión. A la izquierda, extremo de una cabeza de impresión en que pueden verse las agujas. Normalmente se utilizan 5 columnas de 7 agujas (arriba).

sión de estas máquinas, utilizando más agujas o acortando los pasos para que los puntos se solapen. Se llegan a conseguir buenos resultados, pero a expensas de la velocidad y de la duración de la cabeza de impresión.

Para obtener caracteres como los de las máquinas de escribir hace falta una impresora de mar-

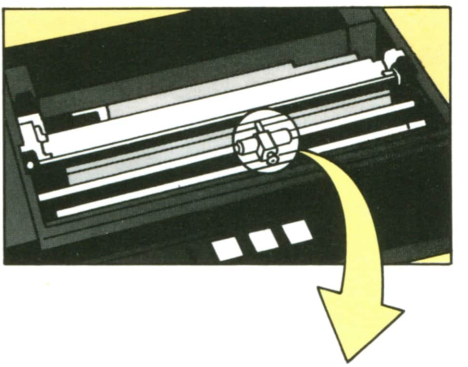
IMPRESORAS ELECTROSTATICAS Y DE CHORRO DE TINTA

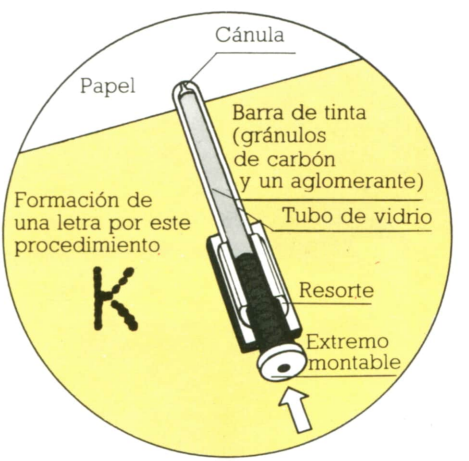
La Timex/Sinclair electrostática, en lugar de configurar los caracteres mediante una columna de electrodos, utiliza dos, de los cuales sólo uno se halla en contacto con el papel, por lo que los caracteres se construyen punto a punto. Para que la velocidad sea mayor, los electrodos van montados en una cinta sin fin, de manera que cuando uno ha barrido el papel, el otro está a punto de comenzar. En las impresoras de chorro de tinta, las gotas de tinta se proyectan electrostáticamente desde la cabeza de impresión al papel. La tinta ha de ser líquida en el momento de salir de la cabeza y secarse un instante después de alcanzar el papel. Olivetti ha resuelto este problema con el chorro de chispa: utiliza una barra de tinta sólida en cuya punta, y mediante una chispa, se funde una pequeñísima gota que es proyectada al papel (página 211).



Estructura del papel. El papel para la Timex/Sinclair lleva un revestimiento de aluminio. Las chispas lo queman descubriendo el soporte negro de debajo.

Mecanismo de cinta sin fin. Lleva dos electrodos montados uno a cada lado de una cinta sin fin: cuando uno termina su barrido da comienzo el del otro.





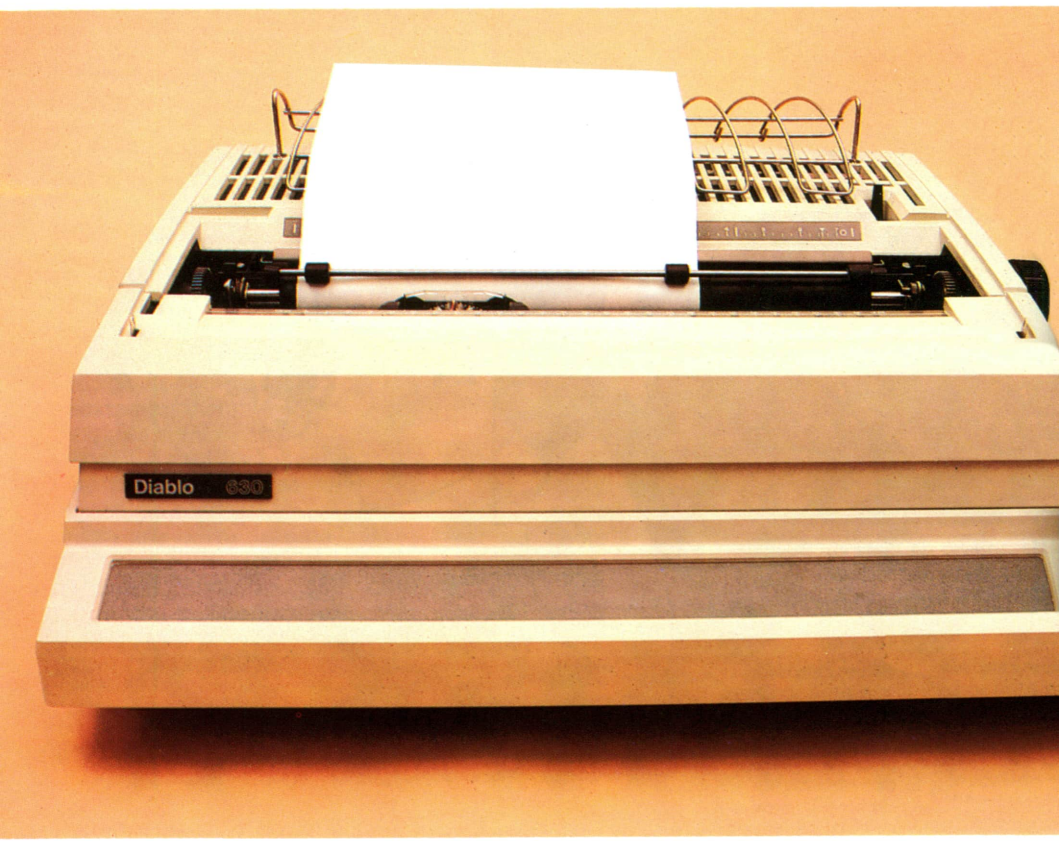
Sección transversal de un chorro de tinta por chispa

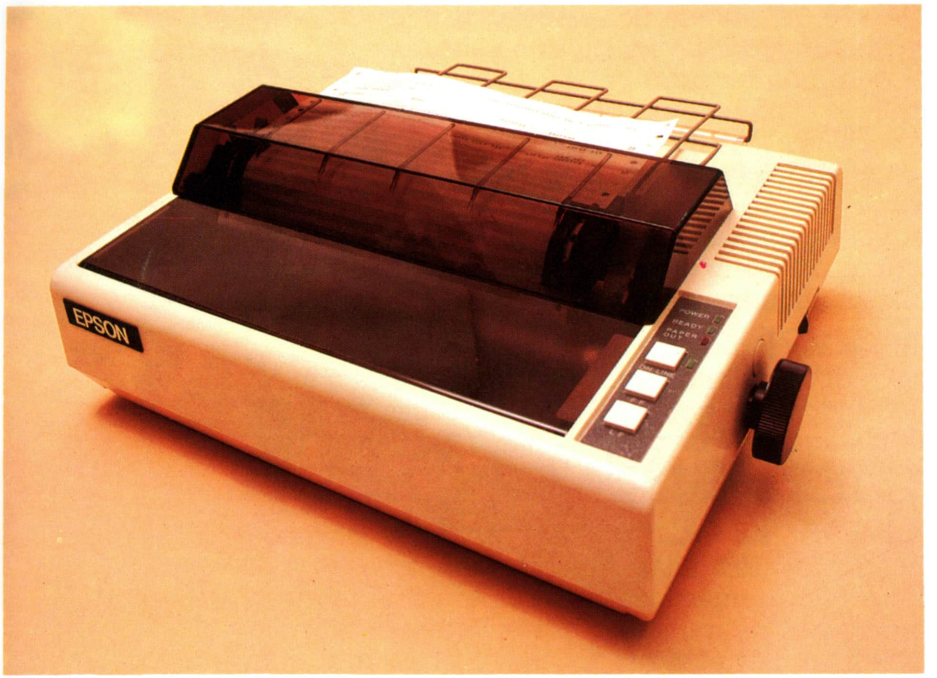
MODELOS DE IMPRESORAS

Existen cuatro tipos corrientes de impresoras. Las de margarita generan una salida impresa análoga a la de las máquinas de escribir. Las matriciales, de bajo costo, son más baratas, silenciosas y veloces, pero la calidad de impresión no es buena. Las térmicas no tienen aplicación en trabajos de calidad, pero apenas hacen ruido. Las impresoras electrostáticas, cuya calidad de impresión no es muy buena, sí son adecuadas para aplicaciones domésticas.

1 Diablo 630. De margarita. Escribe entre 32 y 40 caracteres por segundo. Pesa unos 30 kilos y puede trabajar con hojas sueltas o papel continuo.

1



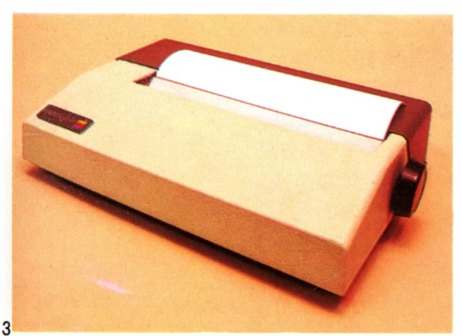


2

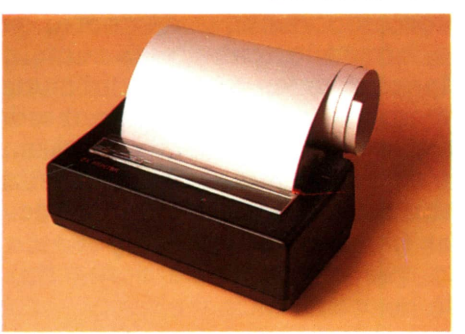
2 Epson MX-80. Matricial con velocidad de 80 caracteres por segundo. Utiliza papel continuo.

3 Apple Silentype. Hasta 40 caracteres por segundo. De tipo térmico, usa papel especial que se vende sólo en rollos.

4 Sinclair ZX. Impresora electrostática sólo utilizable, en la práctica, en las aplicaciones más elementales. Utiliza un papel especial en forma de rollo.



3



4

garita. Posiblemente el lector conocerá las máquinas de oficina modernas, que llevan una rueda enrayada en lugar de palancas o cabezas esféricas. La rueda gira a gran velocidad hasta que queda arriba el carácter deseado, momento en que se activa un martillo electromagnético que lo golpea contra la cinta entintada e imprime el carácter. Las impresoras de margarita son más caras, ruidosas, lentas y pesadas que las matriciales, pero no hay actualmente otro medio de obtener del ordenador una salida impresa de calidad.

Impresoras térmicas y electrostáticas

Otros tipos de impresoras matriciales llevan unos electrodos que generan calor o chispas, en lugar de las agujas y la cinta entintada. Se llaman respectivamente impresoras térmicas o electrostáticas y funcionan con un papel especial (y caro). Suelen ser, sin embargo, más rápidas y silenciosas que las de agujas. Algunos fabricantes de ordenadores domésticos han empezado a fabricar impresoras de muy bajo coste; la Timex/Sinclair ZX Printer, por ejemplo, es una electrostática que forma los caracteres línea a línea utilizando un delgado electrodo o una cinta sin fin.

Nuevas perspectivas

El empeño por lograr una salida impresa de calidad a altas velocidades ha dado sus frutos en nuevas impresoras que están a punto de salir al mercado. En este momento una de las técnicas más prometedoras es la de chorro de tinta, en la que se forman los caracteres sobre el papel proyectándole puntos de tinta diminutos. Están también las impresoras de láser, pero todavía son voluminosas y caras, y apenas están empezando a usarse en los grandes sistemas informáticos. Algunos fabricantes han anunciado modelos «de sobremesa» (llamados así, quizá, por ser del mismo tamaño que una mesa de despacho). Las impresoras de láser son muy rápidas y versátiles, pero no es probable que en el futuro próximo encuentren un sitio en el mercado del ordenador personal.

PERIFERICOS ESPECIALES

Una de las aplicaciones de la informática más en boga se basa en la transferencia de información entre diferentes máquinas a través del sistema telefónico. Para enviar datos por una línea telefónica lo primero es convertirlos en tonos de audio y lanzarlos por una *interface* en serie (véase Módulo 20) a un dispositivo llamado módem, que realiza la verdadera conversión y va conectado directamente al sistema telefónico.

También se puede usar un acoplador acústico. Este aparato lleva dos receptáculos de goma en los que se insertan los extremos del micrófono y el auricular del teléfono, una vez marcado el número y establecida la conexión con el ordenador remoto, equipado también con un acoplador acústico.

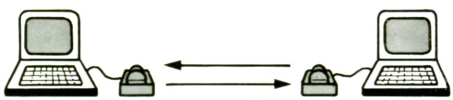
Cada vez son más los ordenadores que disponen de recursos gráficos, pero para sacar copias en papel se necesitan equipos especiales. Lo más común es utilizar un trazador gráfico o *plotter*. Consiste en una pluma controlada por el ordenador que traza el dibujo en papel corriente o en película transparente si se piensa proyectar. Los *plotters* más avanzados llevan ocho o diez plumas, pudiendo generar dibujos a todo color.

Hay también periféricos que reconocen la voz y la sintetizan. A ello se están dedicando en todo el mundo cantidades ingentes de dinero en investigación. Los sintetizadores de voz son más sencillos y suelen basarse en un *chip* de ROM que contiene las versiones digitales de diversas frases, pa-

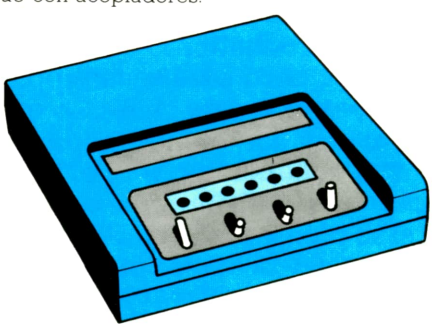
ENLACES ENTRE ORDENADORES

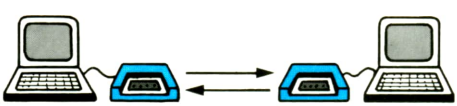
Los ordenadores pueden hablar unos con otros a través del teléfono. Una aplicación de esta posibilidad la constituye el intercambio de datos entre un micro y un gran ordenador. Una gran compañía, por ejemplo, puede tener numerosas sucursales que, todos los días, han de enviar informes sobre transacciones al ordenador de la sede principal. También los usuarios de micros personales pueden utilizar la red telefónica para acceder a bases de datos o intercambiar correo electrónico.

Para conectar un ordenador al sistema telefónico se precisa un «módem» o un «acoplador acústico». Estos dispositivos aceptan datos del ordenador y los convierten en tonos de audio que circulan por las líneas, siendo decodificados en el otro extremo.



El módem. Va directamente cableado al teléfono y conectado al ordenador de la misma forma que el acoplador. Como la conexión entre los ordenadores es directa, a los módems no les afecta el ruido exterior y la transmisión de datos es más rápida que con acopladores.





El acoplador acústico. Este dispositivo (a la derecha) permite conectar el ordenador al sistema telefónico. El acoplador se conecta al ordenador normalmente mediante una *interface* RS232, y lo único preciso es marcar el número de teléfono del ordenador al que nos queremos conectar. Una vez establecida la conexión, se encaja el teléfono en los receptáculos de goma del acoplador. Aunque fáciles de usar y más baratos que un módem, los acopladores son más lentos en recibir y transmitir, y son muy sensibles a los ruidos de fondo.

Altavoz. El acoplador convierte los datos en serie procedentes del ordenador en tonos de audio que luego recoge el micrófono del teléfono.

Micrófono. El acoplador «escucha» las señales procedentes del ordenador remoto en la forma de tonos de audio, las traduce a datos en serie y los envía al ordenador.

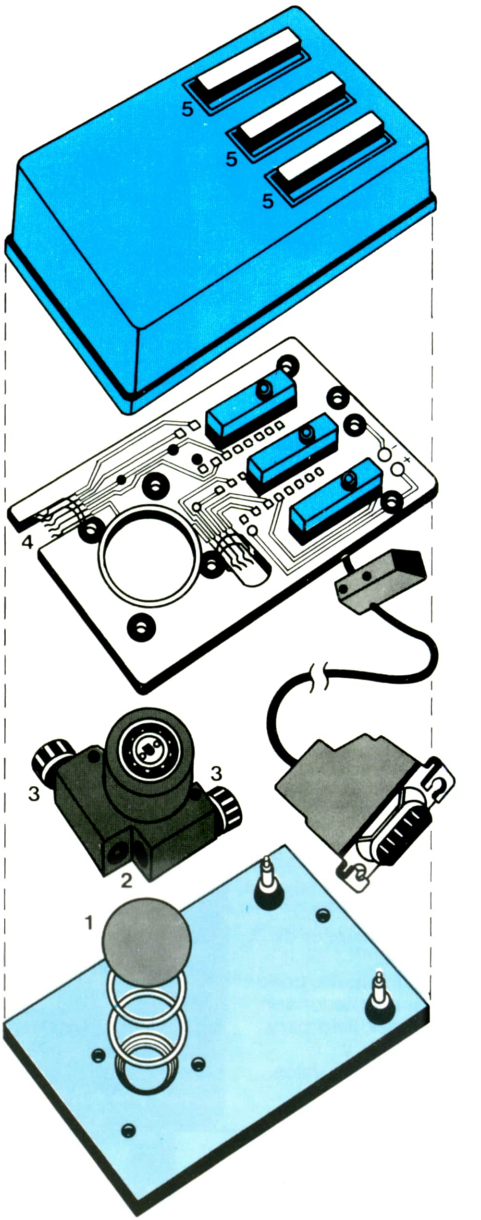
Commutador de línea. Conectado, pone el acoplador «en línea»; listo para recibir y transmitir datos.

Interface **RS232.** A través de esta *interface* se comunica con su ordenador.

Conector de red

EL RATON MECANICO

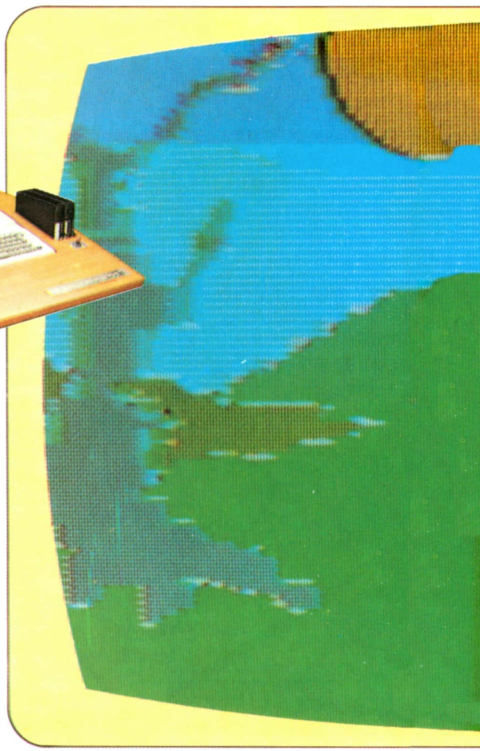
Su propósito es reducir al mínimo el uso del teclado. No es más que una cajita conectada al ordenador por un cable. En la pantalla aparece una flecha que se desplaza siguiendo los movimientos del ratón al arrastrarlo por la mesa. Una vez situada la flecha sobre el símbolo de la tarea elegida, apretamos un botón para que el ordenador se dé por enterado.

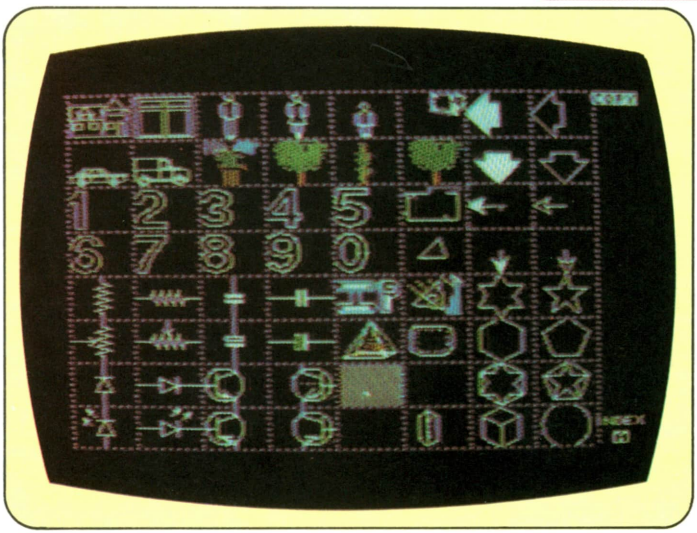


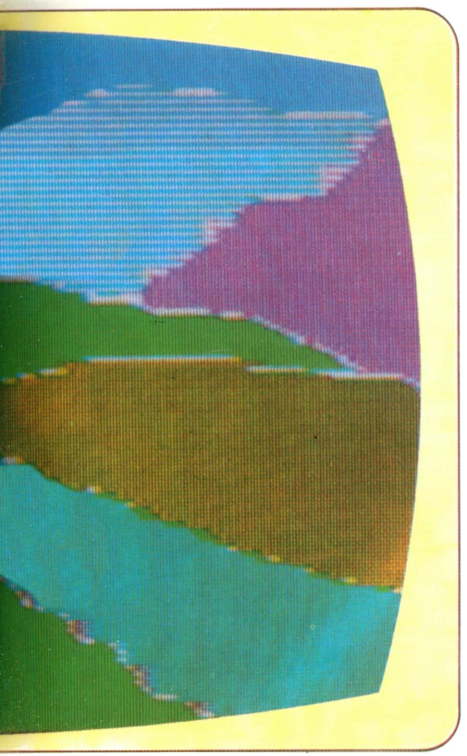
El ratón por dentro. Al moverse el ratón gira una bola de acero (1) y su movimiento se transmite a dos pequeños tambores (2). La bola hace girar los tambores proporcionalmente a la longitud y dirección del desplazamiento. Los extremos de los tambores llevan conectadas unas ruedecitas (3) cubiertas con tiras de material conductor y no conductor. Estas ruedas, al girar, envían señales eléctricas cada vez que rotan los tambores. Unos cables (4) conectados a las ruedas decodifican las señales eléctricas y las envían en forma digital al ordenador. Los tres botones (5) que lleva el ratón en su parte superior sirven para escoger la aplicación de los símbolos de la pantalla.

PERIFERICOS GRAFICOS

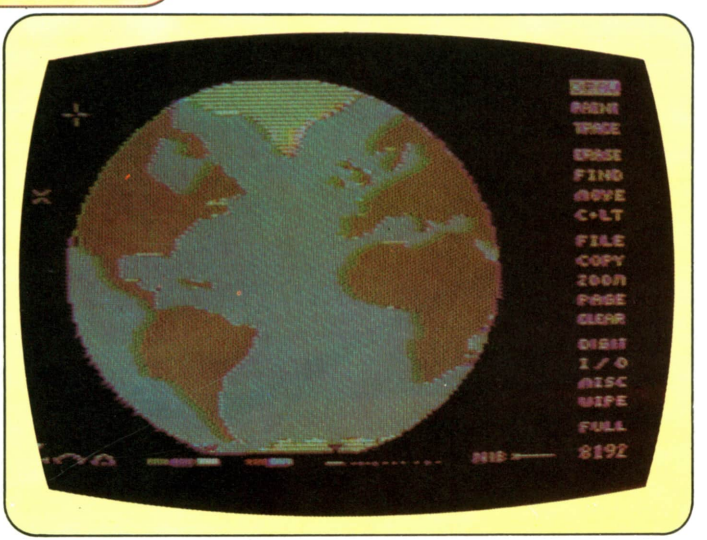
Con un equipo como éste, los artistas y diseñadores pueden crear formas extremadamente complejas de manera muy fácil. Normalmente se usa un *joystick* para «dibujar» en la pantalla. La imagen resultante se puede almacenar en un *floppy* para usarla más adelante o dibujarla en un *plotter*, quizás en color. El sistema aquí ilustrado es el Robocom Bitstik, una amalgama de *software* gráfico especial y periféricos opcionales que convierten al micro —en este caso un Apple II— en una potente herramienta gráfica. El *joystick*, (abajo), permite no sólo dibujar en la pantalla sino también achicar o agrandar los dibujos, girarlos y «posicionarlos» en distintos lugares de la pantalla. De este modo, se puede crear un dibujo extremadamente complejo por partes, reduciendo cada parte y colocándola luego con las demás. Se puede aumentar en cualquier momento el tamaño de una parte de la imagen *(zoom)*, quedando el detalle sólo limitado por la memoria disponible.







Diseño asistido por ordenador. Este «paquete» de diseño permite crear y almacenar en disco bibliotecas enteras de dibujos o de elementos de dibujos (véase a la izquierda). Un sistema de diseño asistido por ordenador típico (página 220), con *software* especial, precisa un ordenador con discos, un monitor, preferiblemente en color, el *joystick* y un *plotter* para sacar copias en papel. En algunos sistemas se puede usar también una impresora matricial.

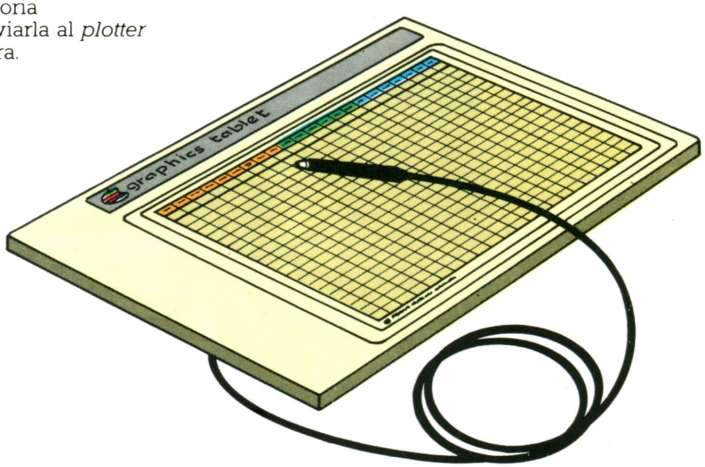


PLOTTERS Y DIGITALIZADORES

Aun cuando podamos manejar gráficos en la pantalla, lo más normal es que, a veces, convenga una copia en papel. Existen impresoras matriciales con posibilidades gráficas, rápidas y baratas, pero de baja calidad de imagen.

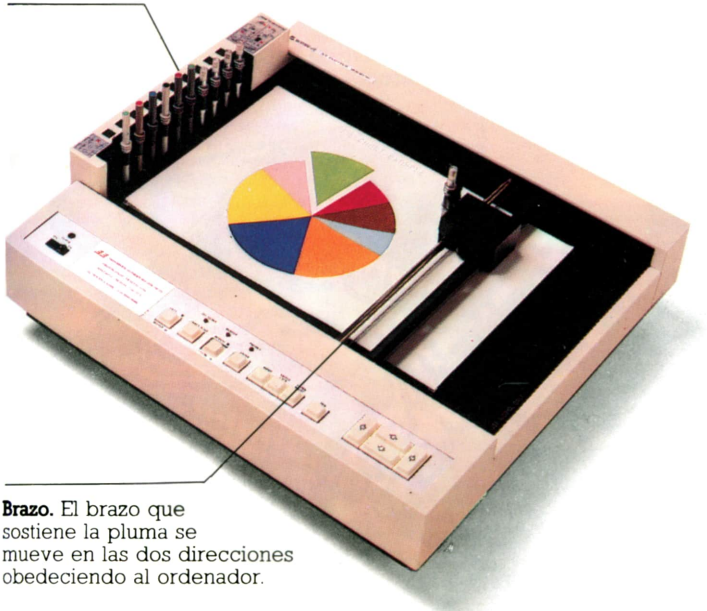
Para trabajos de buena calidad el *plotter* es el ideal. Lleva una pluma especial montada en un brazo que se mueve siguiendo las indicaciones del ordenador, trazando cualquier dibujo en papel. Cuando resulta difícil dibujar, pongamos un coche, desde el teclado, está el digitalizador, sobre el que se dibuja con ayuda de un lápiz a medida que lo desplazamos por la tableta digitalizadora, representando luego en pantalla el dibujo resultante.

Digitalizadores. La tableta gráfica permite dibujar a mano alzada con el ordenador. Este va siguiendo la posición del lápiz en la tableta y registrando el dibujo en memoria para, luego, enviarla al *plotter* o a la impresora.



El *plotter*. En los *plotters* el papel se sujeta electrostáticamente. La pluma se desplaza con gran precisión en respuesta a las instrucciones del ordenador. Una de ellas alza la pluma para llevarla a donde sea. También se pueden escribir letras de diferentes tamaños y tipos de imprenta. Muchos cuentan con varios colores.

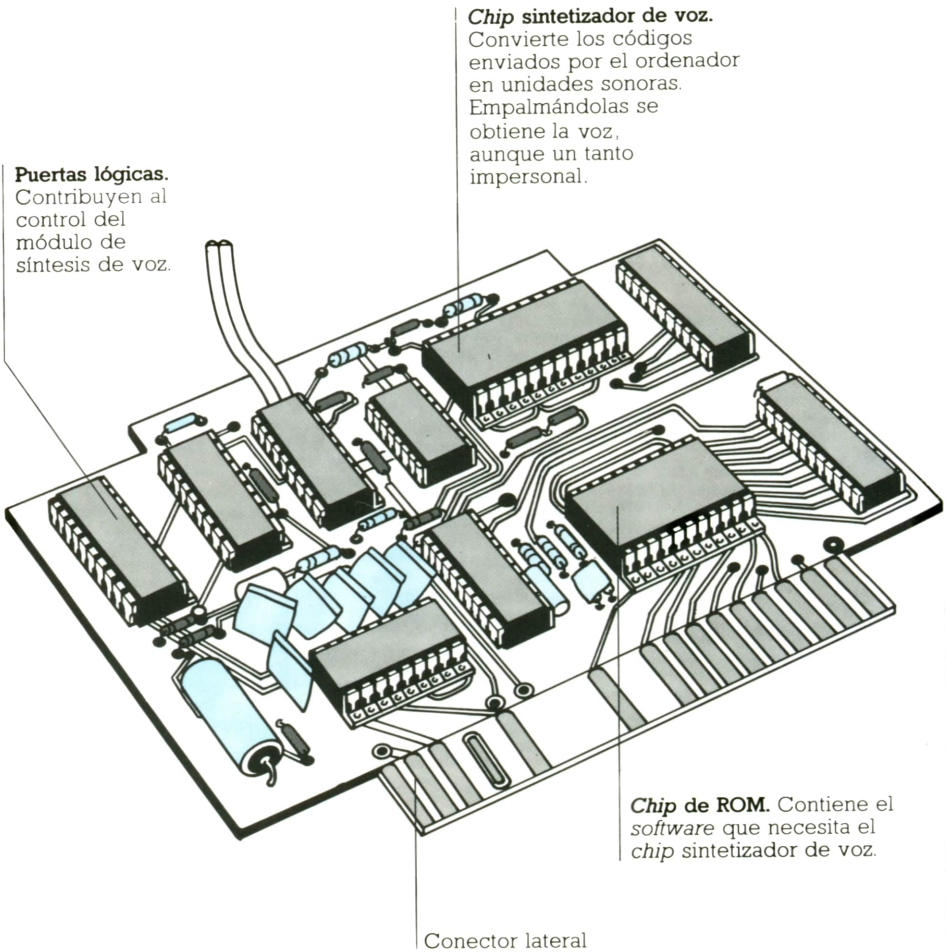
Plumas. Controladas por el micro



Brazo. El brazo que sostiene la pluma se mueve en las dos direcciones obedeciendo al ordenador.

LOS ORDENADORES Y LA SINTESIS DE LA VOZ

Hoy ya es posible que el ordenador hable con palabras sonoras. Las tarjetas con sintetizadores de voz son cada vez más numerosas. Las primeras solían sonar como un astronauta en una alcantarilla, pero el que los micros domésticos posean sintetizadores de voz muy realistas está cerca.



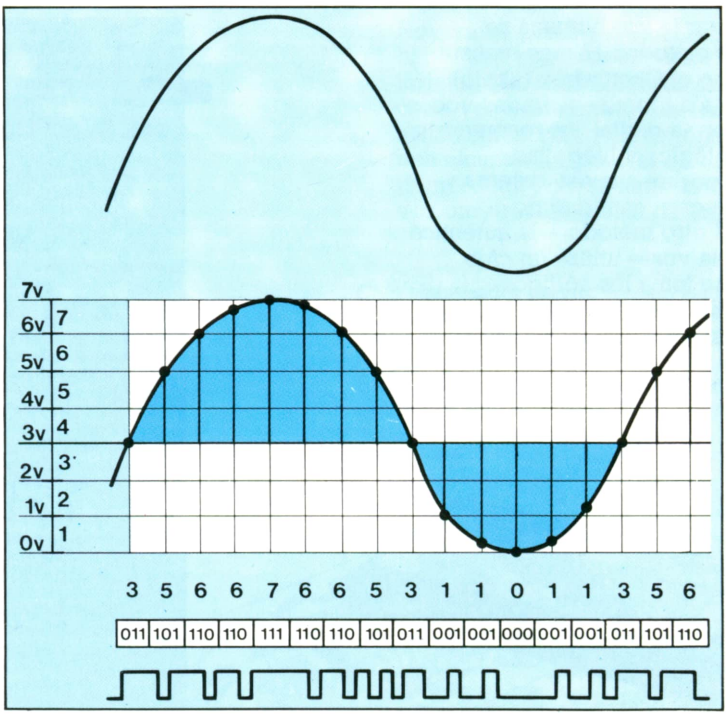
Para sintetizar la voz humana se utilizan dos métodos. El más realista hoy es como el Chatterbox (abajo), que se limita a tener registradas voces reales en forma digital, las reconvierte a forma analógica y luego las reproduce por un altavoz, sistema, que precisa gran cantidad de memoria. El otro método —la auténtica síntesis de la voz— utiliza un *chip* especial que toma los códigos correspondientes a los «alófonos»: 'los sonidos básicos que intervienen en el habla' y los empalma para formar palabras. Son los sistemas del futuro.

CONVIENE RECORDAR

Breves definiciones de los términos aparecidos en esta sección.

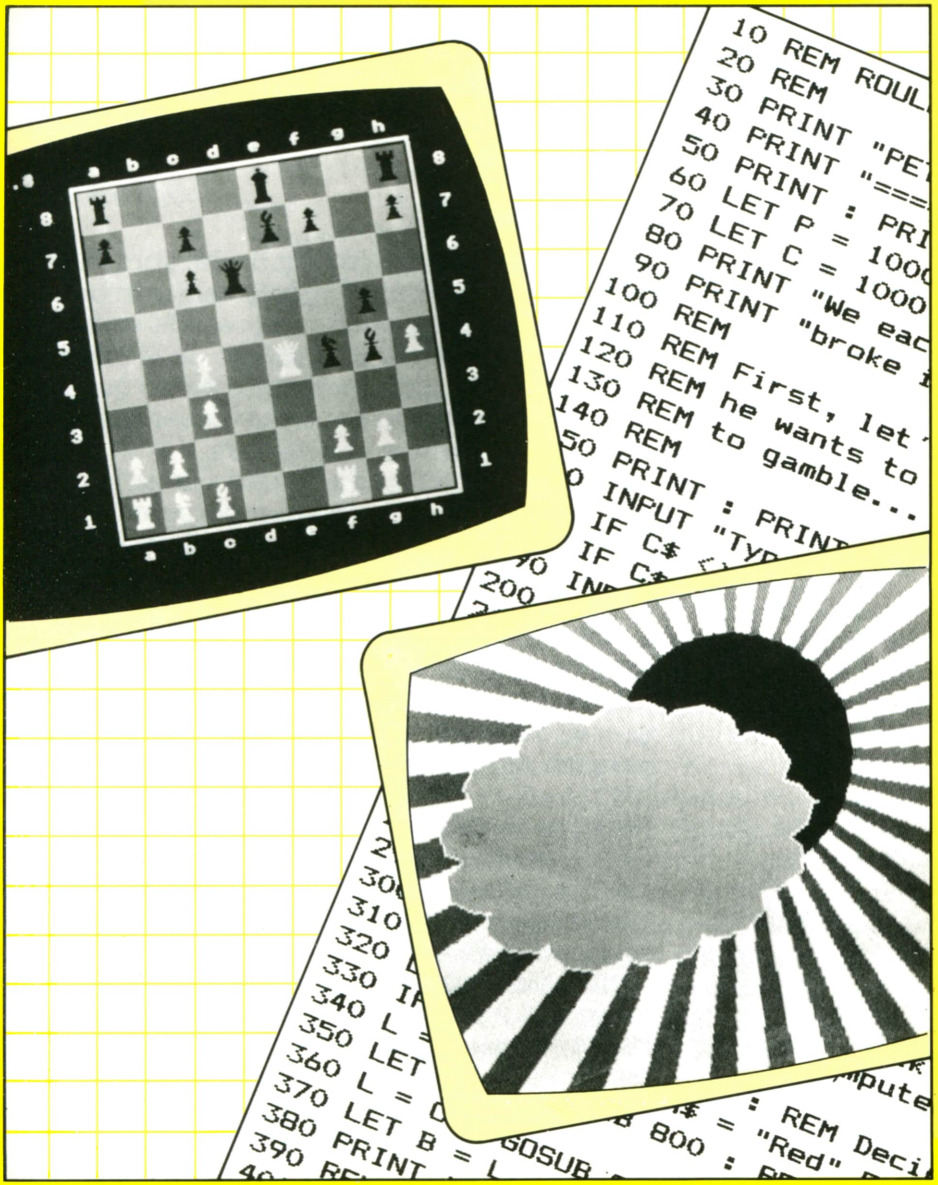
Mapa de bits. Sistema consistente en reservar una zona de memoria para almacenar la información a mostrar en pantalla. A cada bit de la memoria corresponde un punto en la pantalla.
Gráficos. Presentación de la información en forma de dibujos.
Hardware. Conjunto de componentes físicos que constituyen un sistema informático.
Plotter. Periférico para obtener gráficos en papel.
Videodisco. Disco para almacenar información en forma óptica digital.

labras o fragmentos de palabra. Otra aplicación de la microelectrónica es la conversión de señales analógicas en digitales y viceversa. Un conversor analógico-digital mide la señal analógica a intervalos regulares y produce el número binario correspondiente al voltaje de la señal. Para reproducir el sonido se envían los números al mismo ritmo a un conversor digital-analógico, obteniendo la señal analógica correspondiente al original. Como parte integrante de un sistema informático, la conversión A/D y D/A resulta de especial utilidad en ciencia e ingeniería. Permite al ordenador supervisar y analizar muchos procesos de laboratorio y generar señales analógicas para controlar diversos instrumentos.



Conversión analógica-digital. Las ondas sonoras se propagan por el aire en forma de ondas de mayor o menor presión. Un micrófono puede convertir estas ondas en una corriente eléctrica; la corriente resultante varía en consonancia con el sonido, como se muestra en la figura, es decir, es una corriente de voltaje variable. Puede pasarse a forma digital midiendo el voltaje a intervalos predeterminados. Cada vez que se mida se obtendrá un número binario correspondiente al voltaje en ese instante, que puede almacenarse en memoria. Si cambia el voltaje, cambia también el valor de los números binarios que salen del conversor analógico-digital. Introduciendo estos números en un conversor digital-analógico al mismo ritmo que los recogimos, recuperaremos el sonido original en forma de una señal eléctrica que se puede luego reproducir mediante un altavoz.

SOFTWARE


a b c d e f g h
8 7 6 5 4 3 2 1
10 REM ROUL
20 REM
30 PRINT "PE
40 PRINT "===
50 PRINT : PRI
60 LET P = 1000
70 LET C = 1000
80 PRINT "We eac
90 PRINT "broke
100 REM
110 REM First, let
120 REM he wants to
130 REM to gamble...
140 REM
50 PRINT : PRINT
0 INPUT "Typ
IF C$
IF C$
200
310
320
330
340
350 LET
360 L = C
370 LET B = L
380 PRINT
390
800 : REM Deci
= "Red"
GOSUB

INTRODUCCION AL SOFTWARE

Hasta el momento, el contenido del libro se ha centrado principalmente en los sistemas informáticos y en el *hardware*, y sólo de pasada se ha hecho referencia al otro gran apartado de la informática, el *software*, es decir, las instrucciones que indican al ordenador lo que tiene que hacer, y sin el cual no podría funcionar el *hardware*.

A muchos profanos esta parte de la informática puede parecerles un tanto incorpórea y, sin embargo, es una parte vital del sistema. El *software* tiene también su precio; a veces hasta el cincuenta por ciento del coste de un sistema de gestión puede deberse a gastos de *software*. Y lo único que nos dan es un disco y un manual de instrucciones. Sin embargo, no es imprescindible comprar el *software*, pues podemos escribir nuestros propios programas (véase Módulo 27).

Jerarquía del sistema

Para entender las características de los distintos tipos de *software* y su interrelación, conviene representarse el sistema informático (tanto *hardware* como *software)* como una especie de núcleo central rodeado por una serie de capas, como muestra la figura de la página 230. El núcleo es el *hardware* del ordenador, la máquina en sí. La capa externa somos nosotros, los usuarios. Entre ambos se sitúa el *software* que, como puede verse, consta a su vez de varias capas.

La más próxima a la máquina propiamente dicha es la del sistema operativo (véase Módulo 33), que se encarga de las funciones básicas de «buen gobierno» que ha de realizar el sistema. Entre ellas figuran las de ocuparse de las unidades de disco, aceptar los caracteres que llegan del teclado y escribir caracteres en la pantalla.

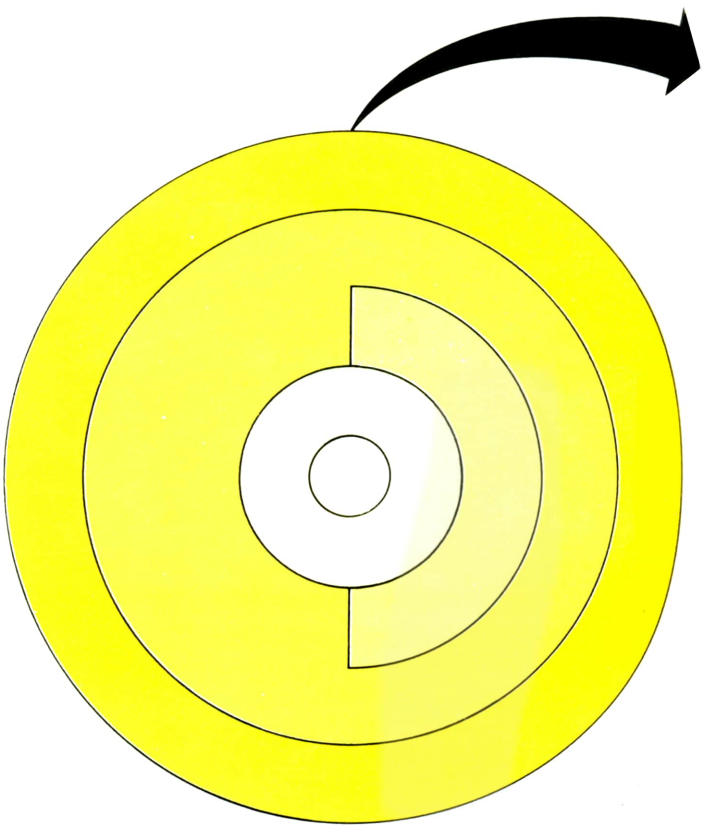
En la mayoría de los micros domésticos el sistema operativo no es perceptible por el usuario por estar incorporado al «intérprete» de Basic —los circuitos que recogen los caracteres tecleados y los traducen a código de máquina inteligible para la CPU (véase Módulo 17). Por el contrario, en la mayor parte de los micros de gestión el sistema operativo es lo primero con lo que se encuentra el usuario al encender su máquina. Normalmente el sistema operativo de un micro de gestión reside en un disco; se enciende la máquina, se inserta el disco y se carga automáticamente el sistema operativo que se encuentra en él. A primera vista esto puede parecer un tanto engorroso: ¿por qué no, por ejemplo, grabar el sistema operativo en ROM? (véase Módulo 19). La respuesta es que, estando en disco, resulta más fácil de modificar, bien para corregir defectos del original, bien para añadir mejoras. Lo único que tiene que hacer el fabricante de *software* es remitir un disco nuevo y el usuario puede pasar al sistema operativo nuevo sin más que cargarlo. El sistema operativo más difundido en los microordenadores de gestión es el CP/M, existiendo para él una amplia gama de *software* (véase Módulo 33).

Ni siquiera un sistema operativo —por complejo que sea— vale por sí mismo de gran cosa. Simplemente nos permite llevar a cabo una serie de trabajos rutinarios tales como «formatear» un disco nuevo (véase Módulo 22) y examinar el «directorio» de un disco. Aparte de esto, nada de utilidad directa puede conseguirse sin más *software*.

En el diagrama de la figura hay otras dos capas de *software:* los lenguajes de programación y los programas de aplicación. Un lenguaje de progra-

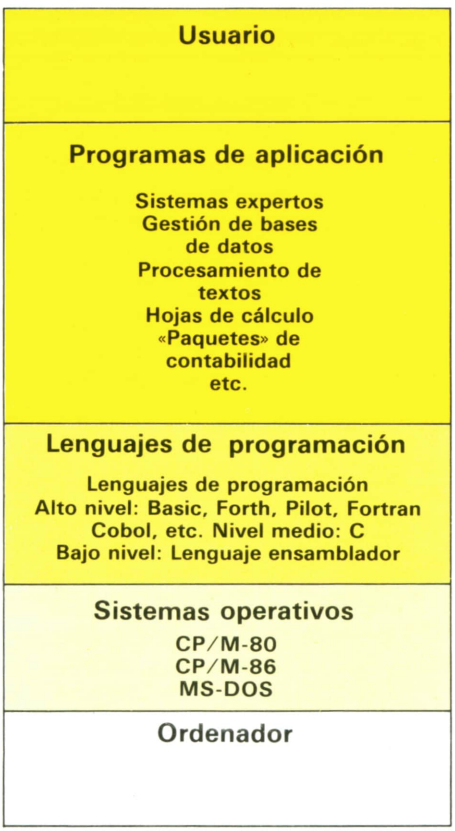
JERARQUIA DEL *SOFTWARE*

Entre el usuario y el *hardware* del ordenador se ubica toda una jerarquía de *software*, estratos de diferentes tipos, cada uno de los cuales desempeña cierto papel. Todo programa ha de servirse de ciertas funciones, tales como recoger del teclado y mostrar en pantalla caracteres o manejar una impresora. Estas funciones están reunidas en un elemento llamado «sistema operativo».



Relaciones entre sistemas. Se muestran las relaciones entre los distintos elementos de un sistema informático y su usuario. En el centro de la figura se halla el *hardware*, el ordenador físico. En torno a él, en círculos, se encuentran el sistema operativo y los programas de aplicación (tabla a la derecha). Entre estos dos están los lenguajes de programación. La capa más externa es el usuario. Estos elementos de *software* permiten la comunicación usuario-micro.

que constituye la capa más interna de la jerarquía del *software*. Las otras clases son: *software* «compilado» (en el que el ordenador, por medio de un compilador, convierte unas órdenes, expresadas en un lenguaje similar al inglés, en instrucciones binarias que se pueden cargar en la memoria y usarse directamente) y *software* «interpretado». En éste ha de haber en la memoria un intérprete del lenguaje que traduzca a código de máquina cada línea del programa en el momento de ir a usarla. Y fuera de este sistema de *software* estamos nosotros, los usuarios.



LENGUAJES DE PROGRAMACION

Tal vez sorprenda que hay multitud de lenguajes de programación. Con el tiempo será posible hablar a los ordenadores en nuestro idioma, pero mientras tanto... La razón de tal diversidad estriba en que, hasta ahora, no se ha ideado un lenguaje capaz de satisfacer las necesidades de cualquier aplicación.

Basic. Es el lenguaje de programación más usado. La mayor parte de las veces se utiliza en la forma de intérprete. Adoptado por casi todos los fabricantes.

C. Es un lenguaje compilado y muy versátil. Genera un código muy compacto y rápido.

mación permite escribir programas de ordenador en un inglés muy simplificado, fácilmente comprensible por los usuarios del sistema. El *software* del lenguaje de programación lo traducirá luego a instrucciones binarias con las que el ordenador pueda trabajar (véase Módulo 13). El lenguaje de programación más popular en el mundo de la microinformática es el llamado Basic. El propósito de los creadores de este lenguaje fue que resultara fácil y rápido de aprender, para que los que pretendan usar el ordenador sin haber recibido una formación en informática puedan familiarizarse pronto con la máquina y sacarle provecho invirtiendo el menor tiempo posible en el aprendizaje de la nueva técnica.

El Basic, sin embargo, no es el único lenguaje de programación que existe. En realidad son muchas las alternativas, aunque en los microordenadores no se disponga de todas. Aun cuando el Basic sea razonablemente bueno en casi todos los aspectos como lenguaje de programación polivalente, adolece de ciertos defectos que lo hacen poco útil para determinados trabajos. Por desgracia no hay ningún lenguaje de programación que pueda ser calificado de perfecto sin paliativos, pese a lo que digan algunos de sus autores. Los lenguajes se diseñaron para una aplicación específica o resultaron ser adecuados para un tipo de programación e inadecuados para otros.

Apropiado para escribir sistemas operativos y *software* de aplicaciones.

Pascal. Otro lenguaje compilado que goza de alta estima entre los programadores de «paquetes» generales de gestión.

Cobol. Buen lenguaje para el manejo de ficheros de información y para «paquetes» de tipo financiero. No muy adecuado en aplicaciones tales como escribir un sistema operativo.

Forth. Lenguaje interactivo que permite al usuario definir nuevas palabras a partir de un conjunto preexistente de palabras definidas.

Logo. Combinando aspectos de la inteligencia artificial y de la psicología, Logo aspira a enseñar a los niños la actitud que deben adoptar para resolver problemas en general.

Pero ni siquiera los lenguajes de programación (salvo pocas excepciones) son un fin en sí mismos, sino medios para conseguir algo. Para llevar a cabo un trabajo práctico se precisa un programa de aplicación, un programa que desempeñe la función concreta para la que en un principio se adquirió el ordenador. El propósito del lenguaje de programación es facilitar y acelerar cuanto sea posible la elaboración de los programas de aplicación (véanse Módulos 34 y 35). A un usuario de gestión, por ejemplo, no le interesarán especialmente los lenguajes de programación: querrá sacar adelante determinado trabajo y su preocupación la constituirá el programa de aplicación capaz de resolverlo. Sin embargo, al aficionado los lenguajes pueden serle de gran interés. Para la mayoría de ellos, *software* de aplicación equivale a juegos (véase Módulo 29), normalmente adquiridos en alguna de las tiendas de *software*. Actualmente se venden juegos magníficos, pero puede resultar enormemente satisfactorio aprender a programar y ser capaces de crear nuestros propios juegos. El Basic suele resultar muy lento para los que precisen movimiento o animación, por lo que el programador de juegos serio tendrá que ensayar ineludiblemente otros métodos de traducir los conceptos del juego en silicio, recurriendo a lenguajes que permitan que sucedan cosas más emocionantes (véase Módulo 31).

Programación de ordenadores

Aprender a programar no es tan difícil ni tan técnico como piensan muchos. Ni siquiera hay que conocer a fondo los ordenadores (aunque, por regla general, cuanto más se sepa mejores serán los programas), y no hay que ser en absoluto un matemático experto. En realidad, los estudios que se han llevado a cabo entre los programadores profesionales demuestran que, si es que existen personas «dotadas» para programar, se trata más bien de las que destacan en idiomas y saben comunicar ideas en un idioma especial (que, como todos los idiomas, tiene sus propias reglas), aunque también la facultad de analizar un problema y descomponerlo en sus partes constitutivas es un factor muy importante.

En el diagrama que veíamos antes se apreciaba que algunos programas de aplicación se sitúan directamente entre el sistema operativo y el usuario, mientras que otros requieren un lenguaje de programación intercalado entre el usuario y el sistema operativo. Hay dos maneras de convertir un programa escrito en un lenguaje de programación en las instrucciones binarias inteligibles por el ordenador. En uno de los métodos, llamado «compilación», el programador ha de escribir su programa con ayuda de un procesador de textos o un editor (véase Módulo 36) y, ejecutando luego un «compilador del lenguaje», traducir el programa a instrucciones binarias. Con un sistema operativo como el CP/M, el programa se puede usar después, sin más que teclear su nombre, pues el sistema operativo busca el programa en el disco, lo carga en la RAM y le «cede» el control. Cuando el programa termina de ejecutarse, el sistema operativo recupera dicho control una vez más.

La mayoría de los ordenadores domésticos y muchos de gestión utilizan un «intérprete» en lugar de un compilador. El intérprete —normalmente para lenguaje Basic— suele residir permanentemente en ROM en los micros domésticos y en disco en los de gestión. Para escribir un progra-

ma con un intérprete Basic, lo primero es cargarlo en RAM (salvo que resida en ROM), del mismo modo que ejecutaríamos un programa que hubiera sido compilado. El intérprete nos permite teclear el programa y guardarlo en *cassette* (véase Módulo 21) o en disco (véase Módulo 22), pero en estas etapas no se traduce a instrucciones binarias. Algunas versiones del Basic se limitan a almacenarlo en RAM en forma de texto ASCII (véase Módulo 13) a medida que lo tecleemos, mientras que otras realizan una especie de operación intermedia que reduce la cantidad de memoria necesaria para almacenar el programa al convertir cada una de las palabras del lenguaje en un símbolo de un solo octeto. Unos pocos intérpretes analizan en esta etapa cada línea tecleada para comprobar que no hemos cometido errores mecanográficos con las palabras propias del lenguaje y que las hemos utilizado correctamente.

Una vez tecleado el programa (o cargado en disco o *cassette)* hay que teclear una orden como RUN para arrancarlo. El intérprete recorre entonces el programa, traduciendo las líneas una por una a los códigos binarios adecuados y ejecutándolos antes de pasar a la línea siguiente. Esto, como vemos, es mucho más lento que ejecutar un programa ya compilado. Dependiendo del ordenador y del lenguaje concreto utilizados, un programa interpretado será entre 5 y 50 veces más lento que su equivalente compilado, aunque en muchas aplicaciones esta falta de velocidad no reviste mayor importancia.

Aplicaciones del *software*

Se dispone hoy de una amplia gama de *software* para casi todas las máquinas, capaz de hacer cualquier cosa, desde juegos a complejísimos trabajos de «inteligencia artificial» (véase Módulo 37).

El mercado de juegos para ordenadores domésticos es inmenso, estimulado por el sinnúmero de máquinas de este tipo vendidas en los últimos años. Se diría que los juegos prefabricados cons-

tituyen una de las aplicaciones principales de los ordenadores domésticos, pese al argumento, tan escuchado, de que se adquieren principalmente por su valor pedagógico. No es infrecuente, sin embargo, que los que entran en el mundo del ordenador doméstico pasen luego a escribir sus juegos propios y originales, o versiones mejoradas de los puestos ya a la venta.

Bibliografía sobre informática

Uno de los problemas con que tropiezan muchos aficionados es la baja calidad de la documentación que acompaña a los microordenadores, en especial a las máquinas de reciente aparición en el mercado. Resulta casi imposible tratar de aprender a programar en Basic con algunos de los manuales que existen actualmente. Aparte de estar mal redactados, con una puntuación desorientadora y una presentación pésima, lo más corriente es que estén plagados de términos técnicos, mal escritos y, a veces, sean completamente inexactos.

Aun cuando los manuales estén bien escritos, sus autores dan frecuentemente por sentado que los lectores conocen la terminología informática básica, lo que de ningún modo tiene por que ser cierto en el caso de los recién llegados al mundillo. Por ello, en esta sección del libro presentamos algunos de estos conceptos y bosquejamos los principios de la programación en Basic. Por desgracia, los aspectos más complejos de la programación de un ordenador doméstico son precisamente los relacionados con las particularidades más atrayentes de la máquina, como el sonido, el color y los gráficos. Como en Basic no hay establecida ninguna norma sobre el control de estas posibilidades, y como varían mucho de una máquina a otra, no hay modo efectivo de explicar detalladamente su uso, aunque sí repasaremos algunos de los principios generales subyacentes.

Una manera de profundizar en el conocimiento de nuestro ordenador es comprar un libro que trate específicamente de esa máquina. Actualmente,

la aparición de un nuevo micro suele ir acompañada de un aluvión de libros y hasta de revistas. Lo mejor es pasar por alto los libros publicados casi simultáneamente con la aparición del ordenador y esperar un poquito. En general, no suele ser bueno comprar una máquina que acaba de salir al mercado. Por ansiosos que estemos de tener en las manos el último grito en micros, vale más esperar unos meses hasta que se hayan resuelto los inevitables errores y problemas de producción y entrega y hayan aparecido también buenos libros y *software*.

Pese a la mayor seriedad de las aplicaciones de los usuarios comerciales, el *software* para los micros de gestión es en muchos aspectos una cosa mucho más simple. Para empezar, ningún usuario comercial que compre un ordenador para fines exclusivamente de gestión debe contemplar siquiera la posibilidad de escribir su propio *software,* a menos que tenga alguna aplicación determinada extraordinariamente peculiar para la que no existan programas prefabricados. Una cosa es escribir *software* por diversión, porque si sale mal no pasa nada. Pero tratar de poner a punto un programa para, pongamos por caso, gestión de almacenes, exigirá mucho tiempo, un duro trabajo y, si las cosas fueran muy mal con el programa una vez en uso, podría tener efectos desastrosos sobre nuestra empresa. Hay ya en el mercado cientos de programas para gestión de almacenes y para casi cualquier otro aspecto de la vida empresarial donde quepa aplicar un ordenador. Terminaremos por convencernos de que es más barato comprar el paquete de *software* que más se aproxime a nuestras necesidades.

Otra de las plagas del mundo de la informática de gestión la constituye la mala documentación, aunque, por fortuna, va poco a poco mejorando. Cada vez son más los fabricantes de *software* que potencian la facilidad de uso de sus productos *(software* «amistoso» o *user-friendly)* u organizan cursos de formación para los usuarios fina-

les. Tratándose de «paquetes» complejos, vale más seguir un curso de formación que abrirse paso trabajosamente por un manual inmenso. En el curso por lo menos aprenderemos a trabajar con lo más elemental del «paquete» y tal vez nos pongan de relieve algunos aspectos que para nosotros hubieran pasado inadvertidos. En un futuro no muy lejano el comprador comercial llegará a una tienda de informática o de equipos de oficina con un problema de gestión y saldrá con una caja de componentes electrónicos capaz de resolver su problema. De vuelta en la oficina, bastará con desempaquetarla, conectarla al sistema existente y comenzar a usarla de inmediato.

LOS ALGORITMOS

La programación de ordenadores tiene un componente de reto intelectual, otro de conocimientos técnicos y otro de talento creativo, aunque hasta que no se ha familiarizado uno con la programación no llega a apreciar la medida en que esta última cualidad es necesaria. Pero, ¿realmente interesa aprender a programar un ordenador con todo el *software* que existe ya?

Para el usuario comercial la respuesta es, casi con toda certeza, no; no es importante si nuestras aplicaciones son de pura gestión, por las razones que ya hemos visto.

En el caso del aficionado ocurre justo lo contrario: escribir nuestros propios programas puede erigirse en uno de los principales atractivos de tener un ordenador.

¿Qué es un algoritmo?

Algunos de los aspectos de la redacción de un programa —una lista de instrucciones— pueden no ser obvios inmediatamente. No cabe esperar, por ejemplo, que la máquina adivine lo que estamos tratando de decirle que haga. Por tanto, tenemos que tener bien claro lo que queremos exactamente de la máquina antes de ponernos a escribir un programa.

Decidido esto, el paso siguiente es pensar cómo lo va a hacer o, en otras palabras, el método para llevar a cabo una operación determinada, y a este método se le llama «algoritmo». Un algoritmo es una descripción formal del modo en que se efec-

túa uno de los pasos de un organigrama (aunque no sea lo mismo que el organigrama); se conocen muchos algoritmos para efectuar toda suerte de operaciones. A menudo se expresan en notación matemática, pero también los hay que pueden describirse con palabras sencillas. Muchos programadores poseen una vasta colección de algoritmos que utilizan frecuentemente en su trabajo.

Los algoritmos en la práctica

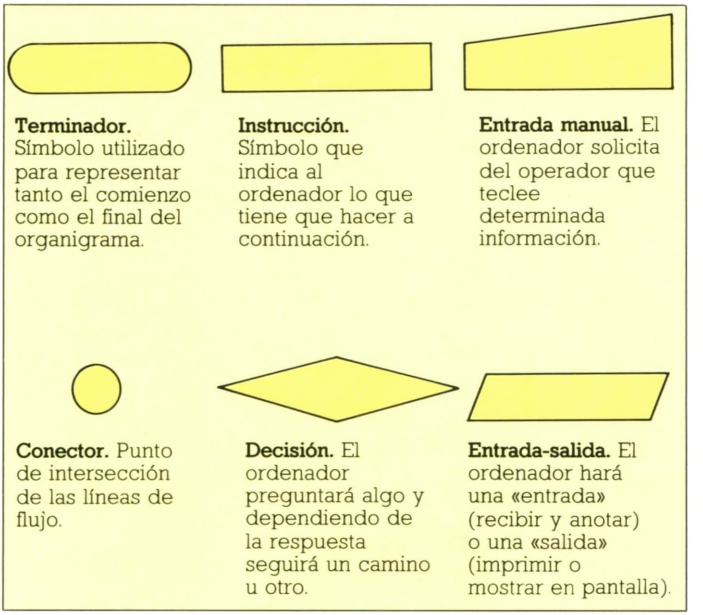
Para aclarar el uso de los algoritmos, tomaremos un ejemplo cotidiano, familiar y ajeno al mundo de la informática: preparar una taza de café. Examinada con detenimiento, esta operación puede descomponerse en una serie de pasos:

1. Llenar de agua un cazo.
2. Ponerlo al fuego.
3. Encender el gas.
4. Echar el café soluble en una taza.
5. Esperar a que el agua empiece a hervir.
6. Verter el agua en la taza.
7. Echar azúcar.
8. Añadir leche.
9. Remover.

Vemos pues que preparar una taza de café consta en realidad de varias etapas diferentes, pero todas ellas podrían a su vez subidividirse. La operación de llenar el cazo, por ejemplo, se descompone en cogerlo, levantar la tapa, acercarlo al grifo, abrir el grifo, etc. Y algunos de estos pasos podrían volver a subdividirse. A veces interesa describir el curso de la acción por medio de una representación visual (véase página 243).

Como vemos en el diagrama, la operación básica de preparar una taza de café coincide con la descripción dada anteriormente, pero se han usado diversos símbolos en diferentes lugares. Hay signos especiales para indicar el principio y el fin del proceso, y cada paso viene descrito en un recuadro independiente, salvo allí donde espera-

SIMBOLOS DE ORGANIGRAMAS



El organigrama constituye una representación visual muy útil del método o «algoritmo» utilizado para resolver un problema. Aunque no sea imprescindible usarlos, existe un conjunto de símbolos normalizados para representar los diversos tipos de operaciones y conviene estar familiarizado con ellos. Son los que se muestran arriba.

mos que el agua empiece a hervir, que está dentro de un rombo, de uno de cuyos lados sale una flecha. Pensándolo un poco, nos daremos cuenta de que «esperar a que el agua empiece a hervir» supone estar mirando constantemente a ver si hierve o no. Mientras sea que no, se sigue esperando, y cuando sea que sí, se salta al paso siguiente. El rombo, por consiguiente, se usa para denotar una operación de toma de decisión, con sali-

das para «sí» y para «no» que indican el curso a seguir en cada caso. Como vimos en secciones anteriores de este libro, la toma de decisiones es algo que los ordenadores saben hacer muy bien.

No es forzoso dibujar uno de estos organigramas cada vez que escribamos un programa, pero, si lo hacemos, conviene saber que existen plantillas para todos los símbolos. En la programación profesional, los algoritmos se usan sobre todo en macroordenadores, en los que un *software* comercial puede estar en uso a lo largo de una década o más y ser preciso modificarlo mucho después de que el autor del programa haya dejado la empresa. Si se cuenta con un organigrama detallado, se facilita mucho el trabajo de comprender lo que ocurre en el programa a cualquiera que trate de modificarlo.

Planificación de un programa

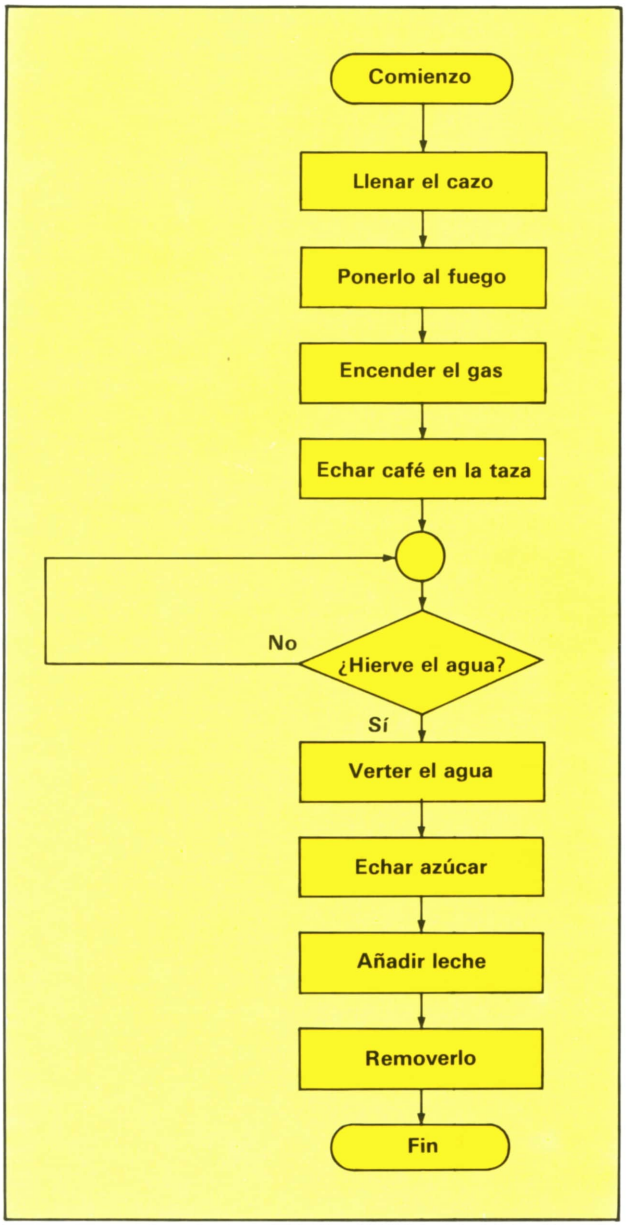
Aunque no siempre sea necesario un organigrama formal, una sesión preparatoria de planificación, incluyendo un análisis detallado de lo que va a hacer el programa y de cómo va a hacerlo es esencial en todo trabajo de programación no trivial, porque aclara las ideas y hace la redacción del código del programa más rápida y menos propensa a errores.

Una manera muy útil de preparar un programa y planificar su curso de acción es redactar una descripción verbal de su cometido, detallando lo que queremos que salga en la pantalla del ordenador, pues en algunos programas, especialmente en los de juegos, ésta puede ser la consideración más importante. Supongamos que se trata de escribir un programa que nos permita teclear una serie de números y calcular su media. Podemos empezar escribiendo una descripción como ésta:

1. Borrar la pantalla.
2. Pedirle al operador que teclee un número.
3. Sumarlo a los números precedentes (si existen).

ORGANIGRAMA PARA PREPARAR CAFE

Casi todos somos capaces de preparar una taza de café, sin analizar cada una de las etapas necesarias. Por trivial que sea el ejemplo, demuestra que los problemas del mundo real se pueden descomponer en sus elementos, primer paso para llegar a una solución informatizada de un problema.



4. Incrementar en uno la cuenta de los números que llevamos tecleados.
5. Preguntar al operador si va a teclar más números.
6. Si la respuesta es sí, volver al paso 2.
7. Si la respuesta es no, dividir la suma entre el total de números tecleados.
8. Mostrar el resultado en la pantalla.

Se evidencia de inmediato que nos hace falta un medio de guardar dos números en la memoria del ordenador: la suma de los números tecleados hasta el momento y el contador del total de números tecleados. En el Módulo 27 veremos detalladamente la manera de hacerlo. Aquí, para no complicar las cosas, pondremos un nombre a estos números y supondremos que el ordenador se encarga del resto. Llamamos SUMA a la suma y CONTADOR al contador de números, y con estos rótulos dibujemos otro organigrama (véase a la derecha).

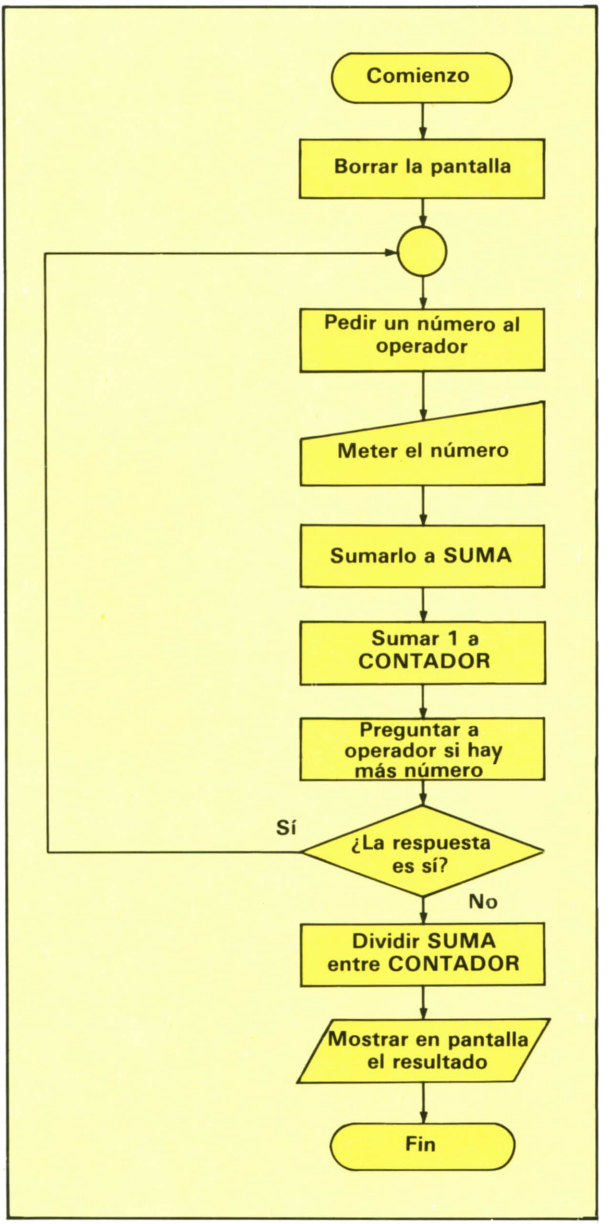
A estas alturas, el lector habrá comprendido ya los principios de la construcción de organigramas y podrá practicarlos, aunque sea sin ordenador, analizando tareas cotidianas que normalmente se efectúan sin pensar.

Ventajas de los algoritmos

Una de las principales ventajas de los algoritmos es que nos evitan recurrir a un lenguaje de programación determinado. Como veremos en el Módulo 32, actualmente son muchos los lenguajes que se emplean para diferentes cometidos, y sería una pena encontrar un método particularmente ingenioso y eficaz de hacer algo pero escrito en un lenguaje distinto del que nosotros usamos. Un algoritmo es «independiente del lenguaje». En otras palabras, una vez conocida la manera de hacer algo, se puede expresar fácilmente en un lenguaje de programación, y a ello está dedicado el módulo siguiente del libro.

ORGANIGRAMA PARA LAS MEDIAS

Aquí vemos las ideas en que se basa la elaboración de un algoritmo para el programa de las medias. Supongamos que tenemos una serie de números cuya media queremos hallar. Lo que hay que hacer es contarlos (CONTADOR) y dividir su suma (SUMA) por su número.



PROGRAMACION EN BASIC

Casi sin excepción, todos los ordenadores personales pueden ser programados en un lenguaje de programación concreto: el Basic (Código de Instrucciones Simbólicas de Usos Múltiples para Principiantes o *Beginners' All-purpose Symbolic Instruction Code).* Muchos ordenadores admiten también otros lenguajes, pero como casi seguro que la máquina del lector tiene alguna versión del Basic, y como el Basic es fácil de aprender, será el lenguaje que usemos en este módulo.

El Basic adolece de un inconveniente un tanto molesto: al igual que muchos de los idiomas ordinarios, presenta diversos «dialectos» o versiones pero, a diferencia de las personas, los ordenadores son incapaces de entender dialectos diferentes. Cada ordenador tiene una versión determinada del Basic y no podrá ejecutar programas escritos para otra. La situación resulta especialmente comprometida en los ordenadores domésticos, pues el *hardware* varía bastante de uno a otro —por ejemplo, los formatos de pantalla o las posibilidades sonoras— y los fabricantes crean sus propias versiones del Basic pensadas ex profeso para aprovechar esas características. Al final de este módulo hay un párrafo dedicado a la adaptación de programas de una máquina a otra.

Por estas diferencias no es posible tratar aquí en detalle el uso de las características más llamativas de los distintos micros, y nos limitaremos a

presentar algunos conceptos elementales de programación para que el lector pueda empezar a escribir programas sencillos, pero utilizables.

Por fortuna existe un «núcleo» del lenguaje Basic común a todas las máquinas con mínimas o nulas variaciones. Y, también por fortuna, es posible hacer muchas cosas con este núcleo, incluso sin usar colores, gráficos, etc. Una vez dominado lo fundamental, las posibilidades más avanzadas las podremos aprender en el manual que acompaña a nuestra máquina.

El Basic en la práctica

Lo que a continuación explicamos se sigue mejor teniendo un ordenador delante, pero aun sin él el lector podrá entender bien los principios y hasta escribir unos cuantos programas, por más que sea preciso reconocer que nada puede sustituir al uso práctico de las cosas. Que haya errores es inevitable, pero nuestros errores en nada dañarán el ordenador, y muchas veces se aprende más tratando de averiguar por qué ha salido algo mal que estudiando indefinidamente la teoría.

Suponiendo que tengamos un ordenador, lo encendemos y tecleamos la palabra HOLA. Pulsamos después la tecla RETURN (en algunas máquinas ENTER o NEW LINE) y el ordenador responderá con un mensaje parecido a SYNTAX ERROR o SN ERROR; desde luego, no nos contestará HOLA. Los errores de sintaxis, significan que el ordenador ha encontrado algo que no entiende. Los lenguajes de ordenador, como los idiomas humanos, poseen una colección de palabras que aportan significados específicos y han de usarse correctamente. Las personas aceptan usos ligeramente erróneos o pequeñas faltas de ortografía en los idiomas humanos porque pueden deducir el significado. Pero con los ordenadores no ocurre lo mismo, y hemos de comunicarnos con ellos en sus propios términos: emplear correctamente sólo las palabras que figuran en el lenguaje de programación.

La razón por la que nos ha salido nuestro primer error de sintaxis es que la palabra HOLA no forma parte del lenguaje Basic. Tecleemos lo siguiente, tal como aparece aquí, sin olvidarnos de pulsar RETURN al final de cada línea.

```
10 PRINT "¿COMO TE LLAMAS?"
20 INPUT A$
30 PRINT "HOLA, ";A$;" MUCHO GUSTO"
40 END
```

Hecho esto, tecleemos LIST y después RETURN. El ordenador listará por pantalla lo que acabamos de introducirle. Si hemos cometido algún error, volvamos a teclear la línea encartada. Repitiendo la orden LIST podremos comprobar que la corrección se ha efectuado. Tecleemos ahora RUN y RETURN —RUN es la orden que indica a la máquina que comience a ejecutar un programa. Si sale otro mensaje de error en la sintaxis lo más probable es que hayamos cometido algún error mecanográfico, y hay que listar de nuevo el programa y comprobarlo. La ? que aparece significa que el ordenador está esperando que le digamos algo, así que tecleemos nuestro nombre y pulsemos RETURN. El ordenador responderá entonces con el saludo programado. Analicemos el funcionamiento del programa para entender lo que ha ocurrido.

Análisis del programa

Lo primero que llama la atención es que las sentencias del programa van numeradas: 10, 20, 30, 40. Todas las sentencias de un programa en Basic deben llevar un número, porque determina el orden en que el ordenador las ejecuta. Los hemos puesto de diez en diez por si en algún momento hay que introducir líneas nuevas en un programa ya tecleado. No importa el orden en que se tecleen las sentencias, con tal de que las numeremos con arreglo al orden en que queremos que se ejecuten. Al teclear LIST, la máquina las ordenará por or-

den de numeración creciente. Una de las ventajas del Basic es la facilidad con que se modifican y corrigen las cosas. Si tecleamos:

```
25 REM ESTA ES UNA SENTENCIA NUEVA
```

podremos comprobar mediante un LIST que el ordenador ha introducido la sentencia nueva entre la 20 y la 30. Por el contrario, si tecleamos un 25 desaparecerá. Teclear el número de una sentencia seguido inmediatamente de RETURN equivale a borrarla. Ahora podemos recorrer el programa sentencia por sentencia.

```
10 PRINT "¿COMO TE LLAMAS?"
```

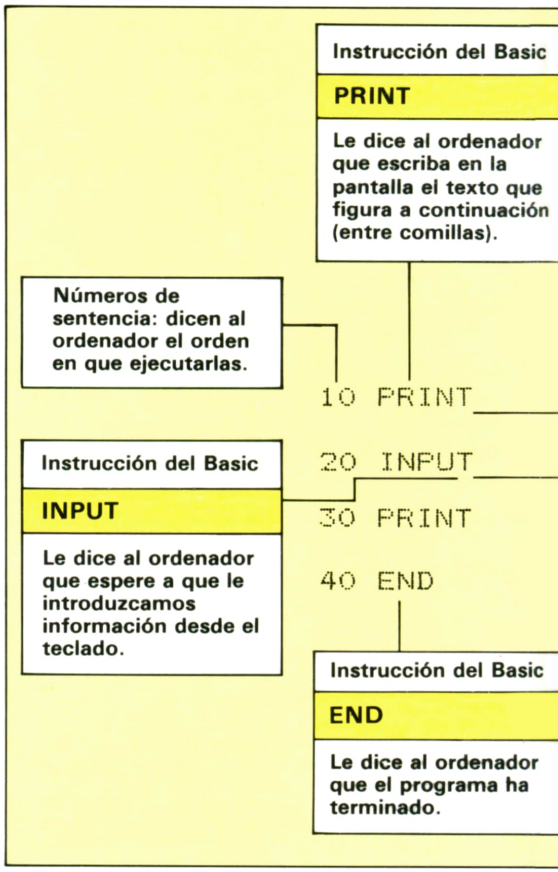
PRINT es la instrucción de Basic que pide al ordenador que escriba algo en la pantalla. Aquí se trata de que escriba: ¿COMO TE LLAMAS?, y se lo indicamos entrecomillando la frase. Así pues, la palabra PRINT seguida de un texto entrecomillado hace que dicho texto aparezca en la pantalla.

```
20 INPUT A$
```

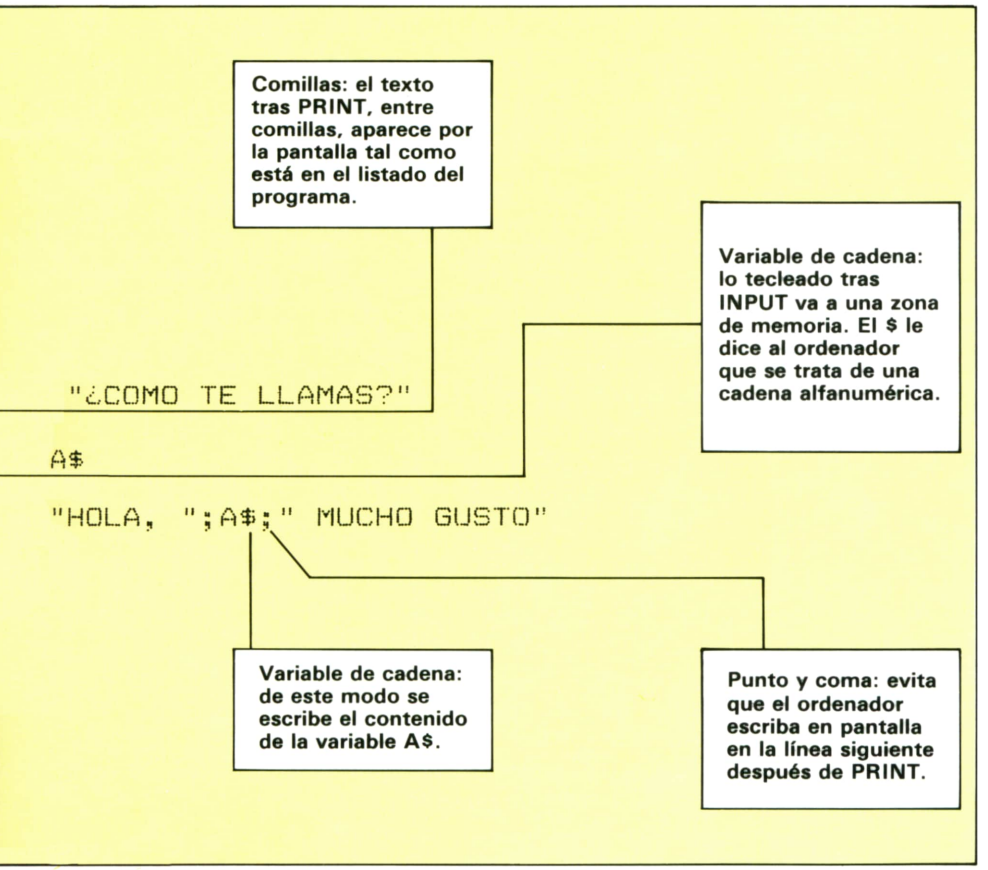
INPUT es la instrucción que le dice al ordenador que en ese momento le vamos a introducir algo desde teclado, y que lo introducido lo va a utilizar el programa, para lo cual ha de tenerlo almacenado en memoria. Más adelante, cuando el programa necesite esa información, el ordenador sabrá encontrarla, así que lo único que tenemos que hacer es reservar una zona de memoria y darle un nombre. A esta zona de memoria reservada se le llama «variable», porque su contenido puede cambiar, y a su nombre se le llama nombre de variable. El nombre de la variable de la sentencia 20

DESGLOSE DE UN PROGRAMA

En el sencillo y breve programa que aquí comentamos se muestra cómo un ordenador escribe información en la pantalla y recibe la procedente del teclado haciendo de ella el uso que el programa determine. En este caso, la máquina nos pregunta nuestro nombre y lo usa para saludarnos. El ejemplo sirve para ilustrar claramente algunos conceptos de la informática, que bien podemos aplicar y perfeccionar luego en nuestros programas.



es A, y la información que se teclee como respuesta a la instrucción INPUT se almacenará en una zona de memoria llamada A. El ordenador ha de saber además el tipo de información que se va a almacenar ahí, si son números o texto. A la variable que contiene números se le llama «numérica» y a la que contiene texto, variable «de cadena» (de cadena de caracteres). Se indica que el tipo de una variable es de «cadena» poniendo el signo $ al final del nombre de la variable. Si no lleva el signo $ se entiende que es numérica. Si volve-



mos a escribir la sentencia 20 omitiendo el $ después de la A y tratamos de teclear nuestro nombre, obtendremos un mensaje de error y la ejecución del programa se detendrá.

Si, estando escrito el programa como en un principio, tecleamos un número en lugar de un nombre, el programa seguirá adelante porque, con una variable de cadena, casi cualquier carácter imprimible —entre ellos los números— será aceptado y almacenado en la zona de memoria bautizada con el nombre de la variable. Pero con ese núme-

ro tecleado no podremos hacer ninguna operación aritmética. A veces esta particularidad resulta de interés, por ejemplo cuando queremos manejar un dato tal como una dirección, en la que figura el número de la calle.

```
30 PRINT "HOLA,";A$;" MUCHO GUSTO"
```

En esta sentencia volvemos a encontrar PRINT, o sea que vamos a decirle al ordenador que escriba algo, y hay un texto entrecomillado que tiene que escribir. Pero esta vez el texto aparece dividido en dos partes y en medio hay otra cosa. Nótense primero los «punto y coma» a ambos lados de A$. Cuando se ejecutó la sentencia 10 aparecieron en una línea las palabras ¿COMO TE LLAMAS? y luego apareció un ? al principio de la línea siguiente. Esto se debió a que una vez ejecutada la instrucción PRINT y escrito el texto, el ordenador salta automáticamente al principio de la siguiente línea si no se le dice lo contrario. Y se le dice precisamente poniendo un punto y coma al final de la sentencia, y fuera de las comillas (si va dentro escribirá el punto y coma también). Por tanto, en esta sentencia le hemos dicho al ordenador que escriba la palabra HOLA, y luego se detenga en la misma línea. Colocando en el medio A$ le decimos que escriba a continuación el contenido de la zona de memoria bautizada con el nombre de variable A$, y que siga en la misma línea, ya que hay un nuevo punto y coma detrás de A$. Luego hay más texto entrecomillado que el ordenador habrá de escribir en pantalla también: MUCHO GUSTO. Si pusiéramos A$ entre comillas, el ordenador se limitaría a escribir A$ y nada más.

```
40 END
```

El significado de la instrucción END es obvio: le decimos al ordenador que el programa termina aquí y debe detenerse. En algunos ordenadores no es preciso poner la sentencia END, pero es preferible incluirla siempre. Más adelante veremos que puede interesarnos en algunos casos que el ordenador no ejecute el programa completo, para lo que puede ponerse una sentencia END en medio del programa real. En las páginas 250-51 hay un listado del programa con una serie de explicaciones sobre lo que hace cada sentencia.

Del algoritmo al programa

Este primer programa nos ha servido para aclarar algunos conceptos en torno a la programación y al lenguaje Basic, pero verdaderamente no hace gran cosa. En el Módulo 26 vimos un algoritmo para tomar un conjunto de números y hallar su media. Pero, ¿cómo pasar del organigrama al programa? Recordemos que se trataba de teclear una serie de números, sumarlos, llevar la cuenta del número de sumandos y, cuando ya no quedasen más, sacar la media dividiendo la suma total por el número de sumandos y mostrar el resultado en pantalla. Al analizar el algoritmo se puso de manifiesto que teníamos que almacenar la suma y el contador en el programa, y por el ejemplo de antes sabemos que para eso podemos emplear variables, una para cada uno. También hará falta una tercera variable para almacenar temporalmente el número tecleado antes de añadirlo a la suma total. Y como tenemos que preguntar si quedan más números, hará falta una variable de cadena para la respuesta. Las variables del programa serán pues:

T para la suma de los números.
C para el contador de los sumandos.
S para almacenar temporalmente cada número antes de sumarlo al total.
A$ para la respuesta a la pregunta ¿ALGUN NUMERO MAS? (el listado del programa se encuentra en las páginas 256-57).

¿Qué significa el listado?

En el listado aparecen varias cosas nuevas. En primer lugar, las sentencias 10 y 20, LET T=0 y LET C=0. Significan exactamente lo que dicen: ponen a cero los valores de las dos variables T y C —y asegurémonos de teclear la cifra 0 y no la letra O, lo mismo que no debe confundirse el número 1 con la ele minúscula. En muchos Basic se puede omitir el LET de este tipo de sentencia.

Sabemos que el nombre de una variable remite a la zona de la memoria en la que está almacenado su valor. Al comienzo del programa no se sabe exactamente lo que hay en esa zona de memoria, que tal vez haya usado otro programa, y por eso hay que cerciorarse de que contiene el valor inicial que necesitamos (cero en este caso).

En la sentencia 30 hay un pequeño truco. En lugar de usar una sentencia PRINT independiente para escribir el texto que indica al operador, lo que ha de hacer lo ponemos en la misma sentencia INPUT con el mensaje entre comillas, seguido de un punto y coma fuera de las comillas y del nombre de la variable en la que va a ser temporalmente almacenado el número. No es necesario poner inicialmente a cero el valor de S, porque el valor que tecleemos en respuesta a la pregunta sustituirá a cualquier otro que pudiera haber en la zona de memoria llamada S.

En la sentencia 40 se suma a T el número que acabamos de teclear. Lo mismo sucede en la 50, donde se incrementa en 1 el valor de C, es decir, la cuenta de los sumandos.

En la sentencia 60 vuelve a usarse INPUT para escribir en pantalla una pregunta y leer la respuesta del teclado. Como esta vez esperamos un texto, el nombre de la variable finaliza con un $. Respondida la pregunta, hay que tomar una decisión en función de ella: volver a leer otro número o calcular y escribir la media. Esto se hace en la sentencia 70 con la instrucción IF...THEN. Si la respuesta es SI, vamos (GOTO) a la sentencia 30. GOTO reorienta el flujo del programa y nos per-

mite, combinada con IF...THEN, tomar decisiones y seguir diferentes alternativas, siendo esta una de las posibilidades que dan más potencia al ordenador. Nótese, sin embargo, que IF...THEN no se usa sólo con GOTO; también se pueden hacer cosas como IF A=B THEN C=D o IF X=Y THEN PRINT «X ES IGUAL A Y».

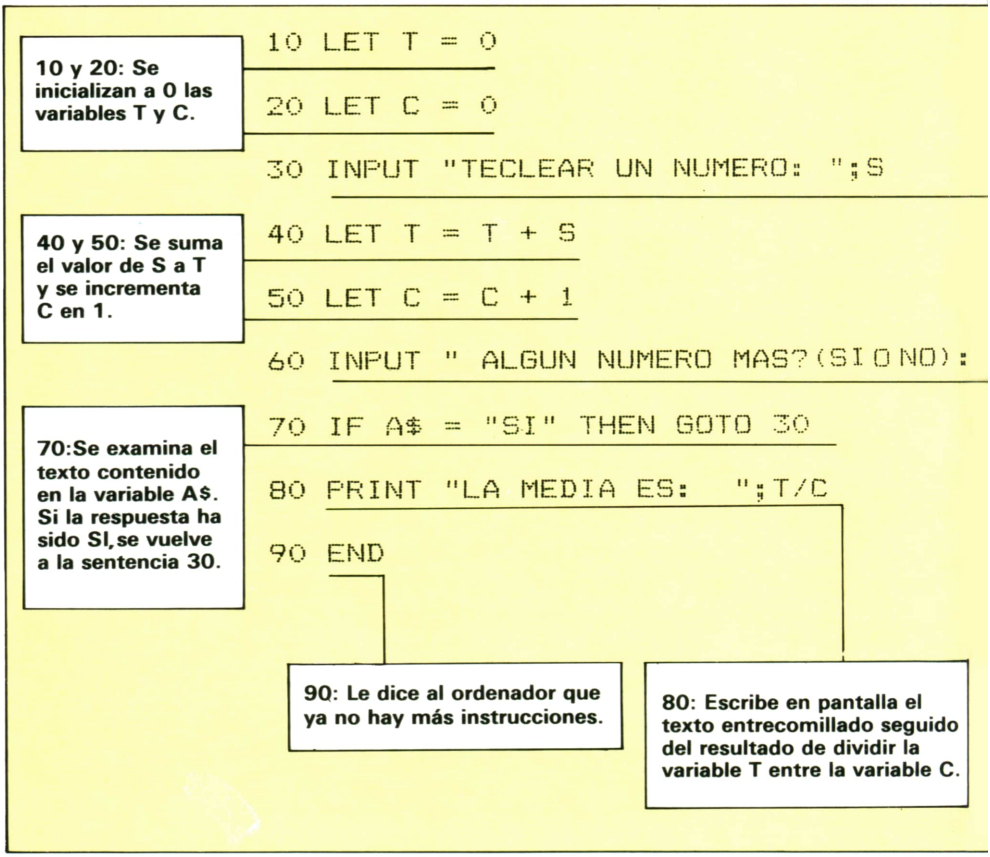
Si se contesta NO a la pregunta, el programa continúa con la sentencia siguiente en lugar de volver a pedir otro número, y en la sentencia 80 escribe por la pantalla la media de los números. Se utiliza PRINT seguido de un texto entrecomillado y luego, en lugar de calcular la media, asignar su valor a otra variable y escribir su valor, ponemos la fórmula que da la media y el ordenador escribirá la respuesta. En este caso la media se calcula dividiendo la suma entre el contador, y para indicarlo se usa el símbolo de la división en Basic: / (un trazo oblicuo). Con esto conseguimos que se escriba el resultado de dividir T (la suma) entre C (el contador). No queda más que el END de la sentencia 90 que pone fin al programa.

Modificación del programa

Supuesto que lo hayamos tecleado exactamente como aparece aquí, el programa funcionará bien en casi todas las máquinas. Como ilustra varios conceptos importantes en informática, conviene practicar introduciendo algunos cambios —quizá distintos mensajes— y observando sus efectos.

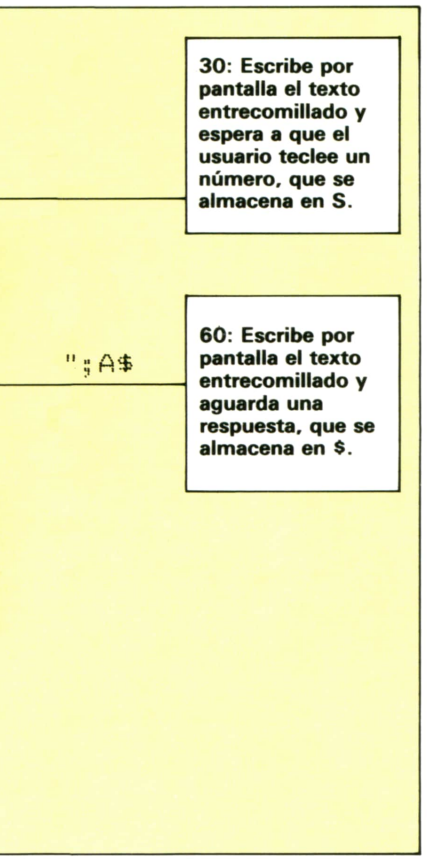
Como programa, sin embargo, deja mucho que desear. Por ejemplo, aburre y es una pérdida de tiempo tener que contestar SI cada vez que queremos teclear un número más. Se podría recurrir a otro criterio para decidir si hay que volver o no a la sentencia 30. Por ejemplo, decidir que la media hay que calcularla cuando se lleven tecleados diez números. Basta para ello suprimir la sentencia 60 y reescribir la 70:

```
70 IF C<>  10 THEN GOTO 30
```



El símbolo <> significa «distinto de». En realidad es la unión de dos símbolos usados con frecuencia en matemáticas: <, que significa menor que, y >, que significa mayor que. Así que lo que la nueva sentencia dice es «si C es menor que o mayor que 10» o, lo que es lo mismo, «si C es distinto de 10». Este programa modificado irá pidiendo números hasta que hayamos tecleado 10 y luego, automáticamente, escribirá la media.

El problema de este procedimiento es que resta versatilidad al programa, pues ya sólo sirve



PROGRAMA DE LA MEDIA

Un sencillo programa que permite calcular la media de una serie de números. Para ello, el programa tiene que llevar la cuenta de la cantidad de números tecleados —lo hace en la variable C— y de su suma —en la variable T— La media la calcula en la sentencia 80, dividiendo T (la suma) entre C (el contador).

Podemos darle todos los números que queramos, pues la máquina nos pregunta después de teclear cada uno si quedan más. Cualquier respuesta que no sea SI desencadena el cálculo de la media.

para sacar la media de diez números exactamente. ¿No sería mejor usar los números mismos como criterio para saber si hay que sumar algún número más?

Podemos convenir que el programa siga pidiendo números hasta que le tecleemos uno especial que reconozca como señal de que se ha terminado la lista y queremos que saque la media. El mejor candidato es el cero, aunque esto nos impedirá utilizarlo como un número más de la lista (lo que en ocasiones puede crear problemas). La ma-

nera más sencilla de hacerlo es introducir una sentencia entre la 30 y la 40:

```
35  IF S = 0 THEN GOTO 80
```

y rectificar la 70:

```
70 GOTO 30
```

Nótese que ahora la decisión se toma en una fase anterior del programa, inmediatamente después de tecleado el número. Si no, el cero con el que se pretende concluir el programa sería sumado al total (lo que no importaría) y contado como un sumando más (lo que falsearía la media obtenida). Haciendo estos cambios y tecleando RUN para ejecutar el programa comprobaremos que podemos introducir cuantos números nos plazca y luego terminar con un 0, tras lo que nos aparecerá la media.

Nuevos añadidos

En algunas aplicaciones puede ser útil almacenar todos los números tecleados y escribirlos de nuevo, o someterlos a cualquier otro tipo de operación aparte de calcular su media. Con el programa tal como está, una vez sumado un número al total se «pierde», porque su valor, contenido en la variable S, será reemplazado por el siguiente número que tecleemos. Una manera de evitarlo sería incluir los números en el propio programa:

```
10 A = 254
20 B = 12.065
30 C = 1
```

Este tipo de asignaciones podría prolongarse indefinidamente, pero resultaría tedioso. Otro método sería incluirlos en el programa mediante la sentencia DATA, de la siguiente manera:

```
10 DATA 254, 12.065, 1, 99, 3465, 121
```

Hay una instrucción especial, READ, que permite acceder a la información contenida en DATA. Es como si el Basic tuviera un «puntero» que apuntase al elemento de la DATA que fuera a leer la sentencia READ. Al comenzar, el puntero señala automáticamente al primer elemento de la DATA. Si la siguiente sentencia del programa es:

```
20 READ X
```

el valor de la variable X pasa a ser el del primer elemento de la DATA (254 en este caso) y el puntero avanza, pasando a señalar el siguiente elemento. Si más adelante hay una sentencia:

```
30 READ Y
```

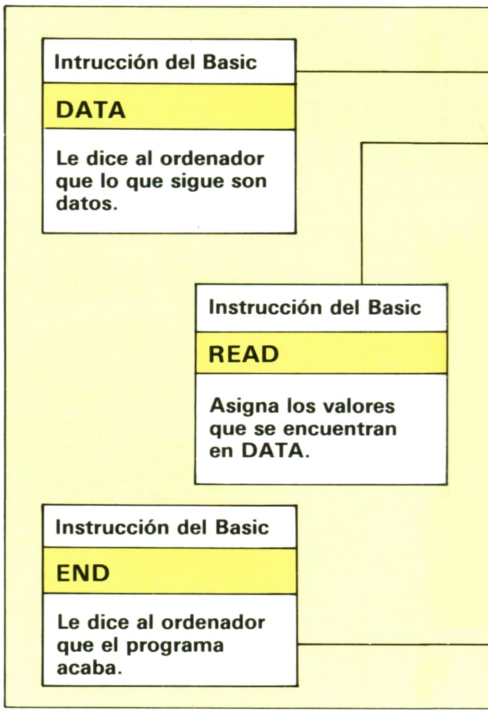
Y tomará el valor del siguiente número de la lista y el puntero avanzará otra posición, y así sucesivamente. Si queremos volver a leer los primeros números, hay una instrucción especial para que el puntero vuelva al primer elemento de la lista, la RESTORE. En las páginas 260 - 61 hay un pequeño programa en el que se hace uso de RESTORE.

La RESTORE de la sentencia 50 vuelve al puntero al comienzo de la lista de la DATA después de leer en las sentencias 20, 30 y 40 los números en las variables A, B y C. Por tanto, las sentencias 60, 70 y 80 leen los mismos valores en las variables X, Y y Z, por lo que X es igual a A, Y igual a B y Z igual a C. Nótese asimismo otra propiedad de la sentencia PRINT: los nombres de las variables aparecen separados por comas, y esto, como puede verse en la pantalla, tiene el efecto de separar los números escritos con algunos espacios.

Pero esto sigue sin ser del todo ideal para nuestro problema de la media. Las sentencias DATA

USO DE LAS SENTENCIAS DATA

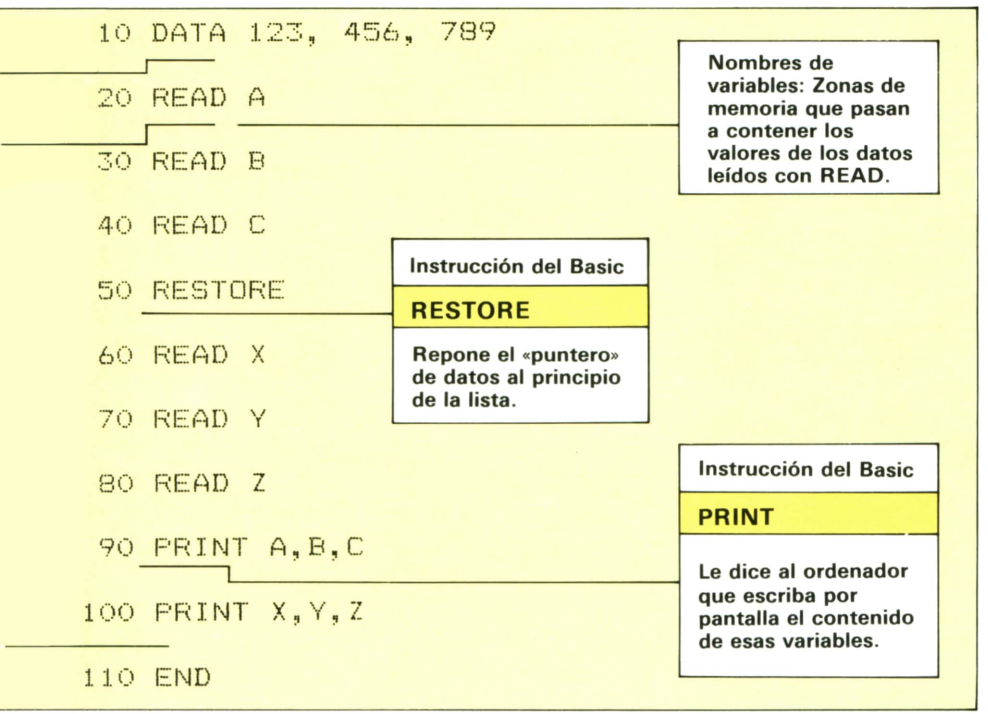
Cuando un programa necesita una serie de datos, se los puede suministrar el usuario a través del teclado o incluyéndolos como parte integrante del programa mismo mediante sentencias DATA. Estas sentencias DATA pueden ir en cualquier parte del programa (aunque conviene agruparlas al principio o al final para que resulten más fáciles de encontrar y modificar si es preciso).



son útiles sobre todo para poner en determinadas variables unos valores que serán más o menos fijos, o para «inicializar» variables al principio del programa. Si luego queremos cambiar algún valor o todos, siempre es más fácil hacerlo modificando una sola sentencia, la DATA.

Variables dimensionadas y bucles

Otra de las características del Basic es que permite crear variables, numéricas o de cadena, dimensionadas. Podemos imaginar estas variables como una hilera de casillas a la que hemos dado un nombre único, pongamos que A. Para usar una de estas variables en el programa hemos de declarar previamente su tamaño mediante la sentencia DIM (de «DIMensión»):




```
10 DIM A(50)
```

Con esto informamos al ordenador de que vamos a usar una variable dimensionada de nombre A y con 50 casillas (1 al 50). Podemos referirnos individualmente a cada casilla de la variable por su número, colocándolo entre paréntesis detrás del nombre, A(1), A(2), A(3), etc. También podemos referirnos a una casilla mediante una variable:

```
10 DIM A(50)
20 X = 2
30 A(X)=123
```

Este programita coloca el valor 123 en la casilla 2 de la variable dimensionada A, porque hemos puesto a 2 la variable X y la hemos usado para decirle al ordenador a qué casilla nos estábamos refiriendo. En esta posibilidad de acceder a las casillas mediante una variable reside la potencia de las variables dimensionadas. Y en Basic las variables dimensionadas suelen ir asociadas a una estructura llamada bucle FOR...NEXT.

Un bucle es un trozo de programa que el ordenador ejecuta reiteradamente. Lo normal es que lo ejecute un número determinado de veces, pues en caso contrario el programa no acabaría nunca. He aquí un ejemplo de bucle:

```
10 PRINT "HOLA- ESTO ES UN BUCLE"
20 GOTO 10
30 END
```

Si ejecutamos este programa, las palabras HOLA-ESTO ES UN BUCLE aparecerán interminablemente en la pantalla y habrá que pulsar la tecla ESCAPE (o BREAK) para detenerlo. Nunca se llegará a la sentencia 30 porqué, tras escribir la frase de la 10 se pasa a la 20, que se limita a remitir de nuevo a la 10.

Otro ejemplo de bucle:

```
10 X = 1
20 PRINT "ESTE ES UN BUCLE DIFERENTE"
30 X =X + 1
40 IF X < 11 THEN GOTO 20
50 END
```

Esta vez aparecerán en pantalla las palabras ESTE ES UN BUCLE DIFERENTE diez veces y luego el programa se detendrá porque, en virtud de la sentencia 40, sólo vuelve a la 20 si X es menor que 11. Si X es igual a 11, pasará a la 50.

La estructura FOR...NEXT dota de más versatili-

CREACION DE UNA VARIABLE DIMENSIONADA

```
 10 DIM A(50) : REM          Define la variable para almacenar 50 números
 20 LET T = 0 : LET X = 0 : REM  Inicializamos a cero estas variables
 30 FOR I = 1 TO 50 : REM  Comienzo del bucle
 40 INPUT "TECLEAR UN NUMERO: " ;S : REM  Toma el número
 50 IF S = 0 THEN GOTO 120 : REM Al encontrar un cero termina esta fase
 60 T = T + S : REM Suma el nuevo número al total
 70 C = C + 1 : REM Contabiliza el número
 80 A(I) = S  : REM Guarda el número en la variable en el lugar I
 90 NEXT I
 00 REM En este punto estamos ante una situación interesante. El bucle está
101 REM preparado para 50 ejecuciones,  porque ésa es la medida de la variable.
102 REM Si se teclean más de 50 números, en la variable no hay espacio
103 REM entonces, si no se teclea el cero antes de llegar a los 50 números, el
104 REM bucle FOR....NEXT debe terminar y el programa debe avisar al usuario
105 REM con un mensaje. Nótese que en la linea 50 se salta este mensaje, porque
106 REM el usuario ha tecleado un 0.
110 PRINT "NO PUEDO ADMITIR MAS NUMEROS!"
120 PRINT "LA MEDIA ES: "; T/C
130 PRINT "Y LOS NUMEROS DADOS SON..."
140 FOR I = 1 TO C : REM Podemos usar el contador C como límite superior del
141 REM bucle; de esta forma no intentaremos escribir elementos de la variable
142 REM que no hayan sido utilizados.
150 PRINT A(I)
160 NEXT I
170 END
```

dad a los bucles. He aquí un programa que hace lo mismo que el anterior:

```
10 FOR X = 1 TO 10
20 PRINT "ESTE ES OTRO BUCLE"
30 NEXT X
40 END
```

El bucle FOR...NEXT comprende las sentencias 10 y 30; todo lo que haya en medio se repetirá cierto número de veces, número que se determina en la sentencia FOR. Aquí se inicializa a 1 la variable X al comienzo del bucle, y luego se ejecutan las sentencias siguientes hasta llegar a la NEXT X. Entonces se compara el valor de X y el valor final fijado en la sentencia 10 por el segmento «TO 10» de la sentencia. Si X es menor que 10, se vuelve al principio del bucle y se suma 1 al valor de X; si no, el programa continúa con la sentencia siguiente a la NEXT X.

En este caso es la variable X la que controla el bucle FOR...NEXT, pero podrá ser cualquier variable numérica (no de cadena). También los otros dos parámetros del bucle, los valores inicial y final, pueden ser variables:

```
100 FOR A = B TO C
```

Es válida siempre que hayamos dado valores a B y C en un lugar anterior al programa. También podemos usar la variable que controla el bucle dentro del mismo. Supongamos que queremos asignar cierto valor a todos los elementos de una variable dimensionada:

```
10 DIM A(20)
20 FOR I = 1 TO 20
30 LET A(I) = 123
40 NEXT I
```

Con esto, todos los elementos de la variable dimensionada A valdrían 123. Accedemos a cada uno de ellos mediante la variable I, comenzando el bucle con I=1 y poniendo I en lugar de un número dentro del paréntesis que determina el elemento a que estamos haciendo referencia. En el segundo recorrido del bucle, el valor de I habrá pasado a ser 2 y accederemos al elemento A(2).

Con lo dicho hasta aquí queda claro que el problema de la media —conservando los números y escribiéndolos, aparte de calcularla— se puede resolver fácilmente con las variables dimensionadas. Sólo existe una limitación: como hay que dimensionar la variable al principio del programa, en ese momento se establece un tope máximo a la cantidad de números que podemos manejar. En la página 263 tenemos un programa que acepta hasta 50 números, halla la media, y la escribe, seguida de todos los números. Hemos añadido algunos comentarios para sacar partido de otras dos posibilidades del Basic, la sentencia REM y la acumulación de sentencias en la misma línea. REM sirve para poner en el programa comentarios *(remarks)* que puedan recordarnos algún detalle (el Basic ignora todo lo que va detrás de la palabra REM en la sentencia). La acumulación de sentencias en la misma línea, permitida por casi todos los Basic, se realiza separando las sentencias con dos puntos y sirve para ahorrar espacio al escribir un programa.

Programa para el juego de la ruleta

Casi todas las versiones del Basic nos permiten generar números aleatorios. Se trata en realidad de números seudoaleatorios, porque el algoritmo que suele emplearse produce una larguísima serie de números, pero llega un momento en que empiezan a repetirse, por lo que no es del todo aleatoria. Sin embargo, basta para escribir un programa para jugar a la ruleta.

Como la ruleta es un juego de azar, hay que generar bastantes números aleatorios. En esta ver-

PROGRAMA DE LA RULETA

```
 10 REM JUEGO DE LA RULETA
 20 REM
 30 PRINT "CASINO DE PEDRO"
 40 PRINT "================"
 50 PRINT : PRINT
 60 LET P = 1000 : REM Dinero inicial del jugador
 70 LET C = 1000 : REM Dinero inicial de la máquina
 80 PRINT "CADA UNO EMPEZAMOS CON 1000 PTAS. EL PRIMERO EN ARRUINARSE ES
 90 PRINT " EL PERDEDOR. ¡BUENA SUERTE!"
100 REM
110 REM En primer lugar le pregunto al jugador a que color y número quiere
120 REM apostar, a continuación cuánto dinero quiere jugar....
130 REM
140 PRINT : PRINT "¿QUIERES EL ROJO O EL NEGRO?";
150 INPUT "PULSA R O N ";C$
160 IF C$ <> "R" AND C$ <> "N" THEN GOTO 150
170 IF C$ = "R" THEN C$ = "ROJO" ELSE C$ = "NEGRO"
180 INPUT "¿A QUE NUMERO, ENTRE EL 0 Y EL 25, QUIERES APOSTAR?";N
190 IF N < 0 OR N > 25 THEN GOTO 180
200 PRINT " TIENES ";P; " PTAS";
210 INPUT "¿CUANTO QUIERES JUGAR? ";A
220 REM
230 REM Vamos a asegurarnos que tiene esa cantidad.......
240 REM
250 IF A > P THEN PRINT "¡NO TIENES SUFICIENTE!" :GOTO 200
260 IF A = P THEN PRINT "VAS A ARRUINARTE. VERDAD?"
270 REM
280 REM Ahora seleccionaremos de forma aleatoria un número, un color y
290 REM la cantidad de dinero que jugará el ordenador.....
300 REM
310 L = 100: GOSUB 800 : REM Decide entre el negro y el rojo
320 IF L < 50 THEN M$ = "ROJO" ELSE M$ = "NEGRO"
330 L = 25 : GOSUB 800 : REM Da el número
340 LET M =L
350 L = C : GOSUB 800 : REM El dinero que queda es el máximo de la apuesta
360 LET B = L
370 PRINT "APUESTO ";B;" PTAS EN ";M$;M
380 REM
390 REM Ahora gira la rueda mientras decido otro número y color
400 REM
410 PRINT " LA RUEDA ESTA GIRANDO....."
420 L = 25 : GOSUB 800 : REM Decide el número
430 X = L
440 L = 100 : GOSUB 800 : REM Ahora el color
450 IF L < 50 THEN X$ = "ROJO" ELSE X$ = "NEGRO"
460 REM
470 REM Este FOR...NEXT bucle no hace nada. Se pone aquí para entretener
480 REM un momento mientras gira la rueda.......
490 REM
500 FOR I = 1 TO 1000
510 NEXT
520 REM
530 REM ¡Imprimir los resultados!
```

sión, el ordenador y nosotros jugamos contra la banca. Decimos lo que apostamos (color y número, aparte del monto de la apuesta) y el resultado lo decide aleatoriamente el ordenador.

```
540 REM
550 PRINT  : PRINT "LA BOLA HA PARADO EN.....";
560 PRINT X$;
570 PRINT X : PRINT
580 REM Ahora decido quién ha ganado....
590 REM
600 IF C$=X$ AND N=X THEN PRINT "TU GANAS!" : P=P+B : GOTO 660
610 IF M$=X$ AND M=X THEN PRINT "YO GANO!" : GOTO 640
620 REM Nadie gana........
640 P = P - A
650 IF P < 1 THEN PRINT"¡AMIGO, TE HAS ARRUINADO!" : GOTO 720
660 C = C - B
670 IF C < 1 THEN PRINT"¡ESTOY ARRUINADO!" : GOTO 720
680 PRINT "AHORA TIENES ";P;" PTAS"
690 PRINT "YO TENGO ";C;" PTAS"
700 INPUT "PULSE RETURN PARA OTRA APUESTA....";G$
710 GOTO 150
720 PRINT "¿QUIERES JUGAR OTRA VEZ?";
730 INPUT "(S O N) ";G$
740 IF G$ = "S" THEN GOTO 60
750 IF G$ = "N" THEN GOTO 730
760 REM
770 END: REM  Ponemos aquí el stop del programa
800 REM A continuación está la subrutina que calcula el número
810 REM aleatorio entre 0 y L. Hay que tener en cuenta que usa
820 REM la función número aleatorio RND(0), y el uso correcto de
830 REM ella puede ser algo diferente en otro ordenador. Si te da
840 REM un mensaje de error, o los resultados no son los esperados
850 REM comprueba en el manual de tu ordenador cuál es la sintaxis
860 REM correcta de la función RND.
870 RANDOMIZE : LET L = INT(RND(0)*L)
880 RETURN
890 REM
900 REM¡Se ha terminado!
```

Habrá que repetir el proceso básico varias veces para tener en cuenta todas las variables, obteniendo un número aleatorio entre el 0 y un límite superior distinto según los casos. Si, por ejem-

plo, estamos determinando la apuesta del ordenador (sentencia 360), el límite superior es la cantidad de dinero que tiene, representada por la variable C. Si queremos saber si la bolita se ha detenido en rojo o en negro, tomamos un límite superior a 100, y será rojo si el número aleatorio resultante es menor que 50 y negro si mayor de 50 (sentencia 460).

En lugar de escribir una y otra vez la función que genera los números aleatorios, podemos usar una «subrutina». Las subrutinas constituyen una herramienta muy útil en programación, porque permiten llevar a cabo la misma acción repetidamente desde distintas partes del programa. En este caso, la subrutina se encuentra en la sentencia 870, y se recurre a ella mediante la instrucción GOSUB 870, cuyo efecto es transferir el flujo del programa a la sentencia 870 y proseguir desde allí su ejecución hasta encontrar una instrucción RETURN. Encontrada esta instrucción, se regresa al punto inmediatamente posterior al GOSUB que llevó a la subrutina. En cualquier punto del programa se puede bifurcar a una subrutina mediante GOSUB y volver al punto de partida con RETURN.

Es costumbre, por atenerse a cierto orden, poner las subrutinas al final del programa, como hemos hecho en éste. Ahora bien, no hay que olvidarse de colocar una sentencia END antes del comienzo de la(s) subrutina(s), como se ha hecho aquí en la 780. En caso contrario, el programa «se meterá» en la subrutina, seguirá ejecutando el código que encuentre y acabará por llegar al RETURN. Pero como entró en la subrutina accidentalmente no tiene ningún sitio donde volver, por lo que el programa concluirá con un mensaje de error tal como: RETURN WITHOUT GOSUB IN LINE... (RETURN sin GOSUB en la sentencia...), lo que al principio puede ocasionar cierta confusión.

Poco más hay que decir sobre el programa de la ruleta, salvo recordar que se puede aprender mucho sobre las sutilezas de la programación experimentando con juegos como éste. A medida

que vayamos familiarizándonos con nuestro ordenador, podremos incorporar nuevos aditamentos, gráficos sobre todo (tal vez la rueda giratoria y la bolita) y posiblemente hasta el ruido de la bolita al rodar si nuestra máquina tiene sonido. El programa aquí presentado puede mejorarse mucho desde el punto de vista estilístico, se ha preferido la sencillez en beneficio de la claridad. Lo único que tiene en común con la ruleta de verdad es que casi nunca se gana.

Adaptación de programas

Como ya se ha dicho, los distintos ordenadores utilizan «dialectos» diferentes del Basic. Esto suele ser motivo de frustración para el aficionado, pues la mayoría de las revistas de microinformática publican listados de programas, y los más interesantes siempre resultan ser para máquinas distintas de la nuestra. Y si tratamos de adaptarlos, encontramos símbolos gráficos extraños, instrucciones misteriosas y mensajes desconcertantes, que ni funcionan en nuestra máquina ni nos dan la menor pista sobre su significado.

Las mayores dificultades se encuentran en las cosas más dependientes de la máquina: el manejo de la pantalla y el sonido. No es tarea sencilla adaptar a otra un programa con gráficos escrito para determinada máquina. Cada una tendrá su número de líneas por pantalla y de caracteres por línea, y además diferente resolución. Habrá ordenadores con buenas herramientas gráficas que tengan instrucciones como FILL o PAINT, y otros que carezcan de ellas, o que las tengan pero las utilicen de forma distinta.Hay máquinas que permiten «ver» desde un programa lo que hay escrito en determinada posición de la pantalla, y otras no. Lo mismo ocurre con el sonido; normalmente hay que enviar una serie de valores a un *chip* de generación de sonido y, salvo que podamos consultar el manual de la otra máquina para averiguar si lleva el mismo *chip* que la nuestra, lo más probable es que fracasemos.

Otro grave problema deriva del uso de PEEK y POKE. Es grande la confusión que siembran entre los novicios en informática, pero en realidad son instrucciones muy sencillas. PEEK permite examinar el contenido de una posición de memoria, normalmente expresada en decimal (no en hexadecimal), y resulta de especial utilidad en la entrada-salida con mapa en memoria (véase Módulo 20) y en las pantallas con mapa en memoria. Así, por ejemplo, X=PEEK(12345) asigna a la variable X el valor que se encuentra en la posición de memoria 12345 (decimal).

POKE hace justo lo contrario. Permite poner cierto valor en una posición de memoria determinada, y se utiliza también para conseguir cambios rápidos en la pantalla en los sistemas con mapa en memoria, o para lanzar algo al puerto de E/S con mapa en memoria. POKE 12345, 100, por ejemplo, colocaría el valor 100 en la posición de memoria 12345. Hay que tener cuidado con esta instrucción, porque es fácil equivocarse de dirección y meterse en la zona de la memoria ocupada por nuestro programa.

Esperemos que en un futuro próximo se reúnan los fabricantes y se pongan de acuerdo en una versión del Basic común. El American National Standards Institute ha aprobado una norma, pero, como aún no está formalmente concluida, pocos fabricantes se han interesado por usarla. Mientras llega ese día, la única forma práctica de adaptar un programa a nuestra máquina es «agenciarse» una copia del manual del otro ordenador y, armados de paciencia, seguir el método de prueba y error.

Para aumentar nuestros conocimientos de Basic

Una de las mejores maneras de aprender a programar es pensar en algo que nos gustaría hacer y tratar de hacerlo con el ordenador. Es posible profundizar más en muchos aspectos del Basic de

CONVIENE RECORDAR

Breves definiciones de varios términos que aparecen en esta sección.

Ensamblador. Programa que traduce a binario las instrucciones en lenguaje de máquina.
Basic. El lenguaje de programación más difundido, lo llevan casi todos los microordenadores.
C. Un lenguaje de programación que permite el control a nivel del lenguaje ensamblador y suele usarse para sistemas operativos.
Cobol. Un lenguaje de programación pensado y utilizado para aplicaciones de gestión.
Fortran. Un lenguaje de programación utilizado principalmente en aplicaciones científicas.
Intérprete. Programa que ejecuta un programa sentencia por sentencia, en lugar de traducirlo todo de una vez a código binario, como hacen los compiladores. La mayoría de los Basic de los micros son intérpretes.
Sistema operativo. Se ocupa de todas las operaciones que aseguran el «buen gobierno» del ordenador.
Pascal. Un lenguaje de alto nivel, usado en aplicaciones muy diversas y en muchos micros. A diferencia de la mayoría de los dialectos del Basic, el Pascal es un lenguaje compilado.

lo que hemos hecho en este breve bosquejo, pero, a causa de las grandes diferencias entre los distintos dialectos del Basic, no resulta factible en este libro.

Si el manual de nuestro ordenador sirve de poco a la hora de explorar los aspectos más avanzados de la máquina, existen libros para casi todos los microordenadores populares que explican sus características con más detalle que el propio manual. Otra fuente de información la constituyen las revistas de microinformática, sobre todo si hay alguna dedicada especialmente a nuestro ordenador. Por último, es muy probable que no lejos de nosotros haya un club de usuarios, y con un poco de suerte, especializado en nuestra marca de ordenador. Asociándonos a él entraremos en contacto con expertos que pueden ayudarnos y con otros novatos con los que compartir nuestras averiguaciones.

DEPURACION DE PROGRAMAS

Un *bug* («error») es un fallo en el *hardware* o en el programa del ordenador. Al proceso de corregirlo se le llama *debugging* o depuración.

¿Dónde puede estar el fallo?

Cuando el ordenador, deja de hacer lo que, a nuestro juicio, debería o simplemente se para, resulta tentador echarle la culpa al *hardware*, pues es la parte del sistema que se puede ver y tocar. Pero el fallo suele estar en el *software*.

Errores los cometen hasta los programadores más experimentados, y cuando se está empezando a programar son casi inevitables. Algunos serán fáciles de encontrar, pero siempre habrá otros que se resistan, y para localizarlos hay que recorrer el listado del programa hasta dar con ellos.

Hay dos tipos de errores de *software:* sintácticos y lógicos. Los más fáciles de encontrar son sin duda los sintácticos, sobre todo porque la mayoría de los ordenadores son capaces de detectarlos y lanzar por pantalla un mensaje adecuado, por ejemplo: «SYNTAX ERROR IN LINE 1240». Algunos ordenadores nos darán tan sólo un código de error que habrá que consultar en el manual.

Errores sintácticos

Por regla general, los errores sintácticos se deben a simples errores mecanográficos (por ejemplo,

escribir PRIMT en lugar de PRINT), y son tan fáciles de rectificar como de cometer. Hay algunos, sin embargo, que, siendo del mismo tipo, no son tan fáciles de detectar: poner «/» (el símbolo de la división) en lugar de «*» (el símbolo de la multiplicación) puede dar lugar a errores curiosos y difíciles de localizar si el fallo se encuentra inmerso en un listado muy largo.

Otro error sintáctico consiste en utilizar incorrectamente una palabra del Basic. Tomemos, por ejemplo, la sentencia 870 del programa de la ruleta del módulo anterior (véase Módulo 27):

```
LET L = INT (RND(0) * L)
```

En las expresiones de este tipo es muy fácil teclear mal los paréntesis, bien olvidando alguno, bien colocándolo donde no es. Esta sentencia:

```
3520 LET X = INT (SQR(Y/(3*X)))*3:IF
     X=0 THEN GOSUB 4000 GOTO 4420
```

provocará la aparición en pantalla del mensaje SYNTAX ERROR IN 3520, y lo más probable es que perdamos mucho tiempo repasando los paréntesis antes de darnos cuenta de que faltan los dos puntos antes de la instrucción GOTO 4420.

«Caer» inadvertidamente en una subrutina puede originar el caos. Es fácil que ocurra, bien por haber olvidado poner la sentencia END delante de un grupo de subrutinas al final del listado, bien, lo que todavía es peor, por haber olvidado poner la sentencia RETURN en una subrutina. Esto ocasiona que la ejecución prosiga, se entre en la siguiente subrutina, y sólo se regrese al encontrar el RETURN de ésta, momento en el cual han podido ya ocurrir las cosas más insospechadas.

Cuando se está escribiendo un programa complejo, lo más seguro es que haya que añadir y qui-

tar sentencias por todo el listado. En el curso de este proceso resulta increíblemente fácil borrar una sentencia a la que hemos hecho referencia en otro lugar mediante un GOTO o un GOSUB. Al ejecutar el programa y darse cuenta el ordenador de que tiene que ir a una sentencia que no existe nos lo advertirá, pero no deja de resultar molesto. Por ejemplo:

```
120 IF X = INT(Y) THEN GOTO 500
      _
      _
      _
500 REM Esto es un comentario
510 LET Y = SIN(J)
etc.
```

Si tenemos un programa grande y se nos está acabando la memoria, un método para ganar espacio es borrar todas las sentencias REM. En este caso, mientras exista la sentencia 500 no importa que desde la 120 vayamos a ella, pues el ordenador ignora las sentencias que comienzan por REM y el programa continuará con la ejecución de la 510. Pero si con todos los REM borramos la 500, obtendremos un mensaje de error al tratar de dirigirnos a ella con el GOTO.

La mayor parte de los Basic dan una información adecuada sobre los errores sintácticos.

Errores lógicos

Los errores lógicos son aquellos en que la sintaxis es perfecta, pero le hemos dicho a la máquina que haga algo que no es correcto. Un ejemplo sencillo de este tipo de error: queremos saber el valor de la variable X$ y pedimos al ordenador que escriba el de X.

Resulta enormemente frustrante saber lo que queremos que haga el ordenador, estar convencidos de haberlo programado correctamente y ver que la máquina no lo hace bien. El ordenador, obviamente, no puede adivinar lo que nosotros pre-

tendemos. Se limita a seguir las instrucciones que le hemos dado y —supuesto que el programa no contenga errores sintácticos— hará exactamente lo que le hemos dicho. Si le hemos dado un programa con errores lógicos pero de perfecta sintaxis, lo ejecutará, aunque desde nuestro punto de vista el resultado no sea correcto.

Hay varias formas de resolver este problema, aunque ninguna necesariamente fácil. Las líneas con varias sentencias, por ejemplo, pueden causar problemas. Sea:

```
1730 LET X = Y: IF X < Z THEN
X = Z: GOTO 1500
```

En muchas versiones del Basic, la última sentencia de la línea se ejecutará sólo si la anterior (IF X=Z THEN X=Z) es cierta. Si lo que queríamos era ir a la 1500 independientemente de que X fuera menor o no que Z, pero resulta que X es igual o mayor que Z, no pasaremos a la 1500. En este caso sería más seguro escribir:

```
1730 LET X = Y : IF X < Z THEN X = Z
1740 GOTO 1500
```

Esto vale si lo que queremos es ir a la 1500. Podemos averiguar cuál es el comportamiento de nuestro Basic (si es que admite varias sentencias en la misma línea) con este programita:

```
10 A = 1
20 B = 2
30 IF A = B THEN C$ = "NO": GOTO 50
40 C$ = "SI"
50 PRINT C$
60 END
```

Si al ejecutarlo sale en pantalla un «SI», es que en nuestro Basic se puede caer en esa trampa. Si sale un «NO», significa que el resto de las sentencias posteriores a la condicional se ejecutan con independencia del resultado de esa condición.

El otro gran problema es cerciorarse de que las variables tienen los valores que se supone deben tener a lo largo de la ejecución del programa. Normalmente puede comprobarse insertando sentencias que escriban el valor de las variables sospechosas en las partes donde quepa esperar problemas. También podemos insertar sentencias STOP y END en lugares estratégicos.

Puede resultar difícil averiguar si el ordenador ejecuta realmente determinada parte del programa. Algunas máquinas disponen de la instrucción TRACE. Tecleando TRACE ON o TRON antes de ejecutar el programa podremos ver en pantalla los números de las sentencias por donde va pasando. Normalmente, si el Basic lo permite, interesa más poner TRON y su contrario TROFF delante y detrás de la zona por donde sospechamos que no se pasa. Si no aparece en pantalla ningún número de línea, es que en efecto esa parte no se ejecuta. Si no disponemos de la instrucción TRACE, podemos simularla insertando mensajes al comienzo de cada sección del programa: PRINT "ENTRAMOS EN LA SUBRUTINA DE NUMEROS ALEATORIOS", por ejemplo.

El proceso de depuración se facilita si cuidamos el diseño y la organización de nuestros programas. Empecemos por dibujar un organigrama (véase Módulo 26), en el que el programa quede dividido en secciones funcionales y se describa lo que se pretende en cada una. Y cuidemos el diseño de las salidas por pantalla. Al escribir el programa, conviene separar estas secciones lo más posible, dejando unas líneas en blanco con la sentencia REM. Y pongamos al comienzo de cada sección una o más REM, explicando brevemente lo que hace la sección y qué variables usa o cuáles afecta en otras partes del programa.

JUEGOS DE ORDENADOR

En los primeros tiempos de la informática, los ordenadores eran demasiado escasos y caros de mantener para poder dedicarlos a cosas tan poco serias como los juegos, o al menos eso era lo que se decía oficialmente. Los juegos de aquellos grandes ordenadores resultaban descorazonadoramente aburridos en comparación con los de los micros de hoy. Para empezar, las terminales de macroordenadores usadas por programadores y operadores carecían (y esta situación apenas ha cambiado) de posibilidades gráficas y de color, dos cosas que rara vez faltan en las máquinas domésticas.

El *boom* del micro

Cuando empezaron a popularizarse los ordenadores, cada vez fueron más los que se dedicaron a escribir juegos y además se fue dotando a las máquinas de recursos tales como el color, el sonido y los gráficos, que podían explotarse en los juegos y en otros tipos de *software*.

Los juegos de ordenador constituyen hoy un importante segmento de la industria del micro. Algunos de los mayores fabricantes de microordenadores deben su misma existencia a los juegos: Atari, por ejemplo, se dio a conocer con juegos de vídeo y *arcade*, pasando luego a fabricar unos ordenadores domésticos con espléndidas posibilidades gráficas.

Programación de juegos

¿Quién escribe todos estos juegos? Muchos se deben a una serie de jóvenes de talento que trabajan por libre, y cuyos ingresos económicos pueden ser notables. Pero, también hay grandes empresas con departamentos dedicados específicamente a escribir juegos nuevos.

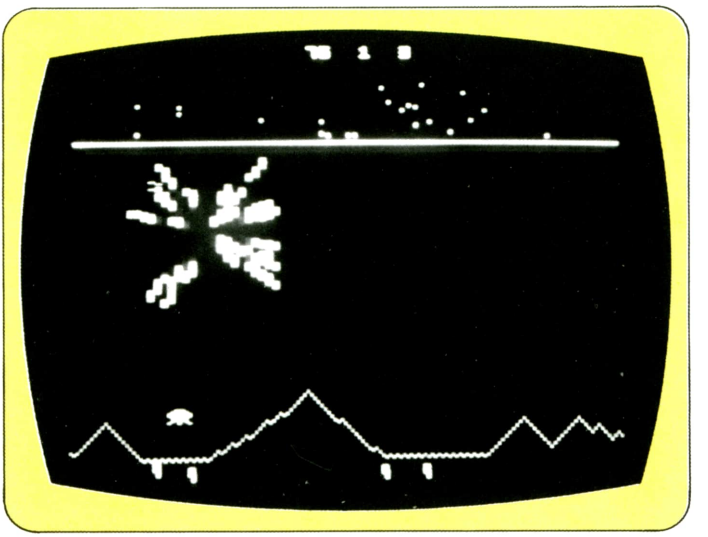
Una vez trazado un bosquejo del juego, unos artistas profesionales diseñan los «escenarios», unos dibujos animados que reflejan lo que debe aparecer en pantalla en cada fase del juego. Al mismo tiempo se refinan y concluyen las reglas y el organigrama. Un equipo de programadores se encarga luego de convertir los escenarios en gráficos y el organigrama en un programa. Las primeras versiones del juego se someten a rigurosas comprobaciones, no sólo para ver si es bueno, sino para cerciorarse de que no hay fallos en el programa,ensayando todas las combinaciones de circunstancias que puedan darse.

El procedimiento de comprobación reviste especial importancia en el caso de los juegos que se venden en cartucho, pues esto supone la fabricación ex profeso de miles de *chips* de ROM.

Tipos de juegos

Los juegos de ordenador pueden clasificarse en tres amplios apartados: los de aventuras, los de «tiro al blanco» (en los que se conceden puntos por destruir al enemigo) y los basados en juegos de mesa tradicionales.

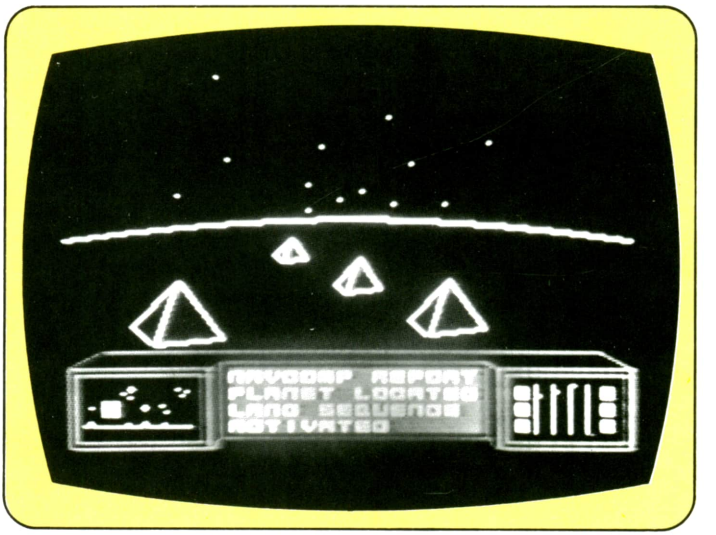
En los de aventuras se trata normalmente de recorrer un laberinto de cavernas y túneles, o algo parecido, equipado con distintos objetos que podemos utilizar o desechar según convenga. El propósito suele ser llegar a cierto punto, sorteando monstruos y pozos sin fondo, y luego hacer el camino de regreso. Todos los objetos que se encuentran servirán para algo, aunque de momento no se sepa bien para qué. Puede ocurrir, por ejemplo, que la cuerda que hemos tirado hace media hora sea justo lo que necesitamos ahora para sal-



Space Invaders. Todo empezó con Space Invaders, hileras de siniestras naves invasoras que descienden lentamente por la pantalla mientras el jugador, a los mandos de un cañón situado en la parte inferior de la misma, trata de destruirlas antes de que aterricen. Existen actualmente innumerables variantes de este tipo de juego, que sacan un buen partido de los gráficos en color rápidos y, en ocasiones, del sonido con que cuentan la mayor parte de los ordenadores domésticos. Aunque la calidad no es tan buena como la de las máquinas de los salones recreativos, las versiones domésticas están mejorando por momentos. El planteamiento de estos juegos casi siempre es el mismo: unos anónimos e indeseables extraterrestres se lanzan hacia nosotros en oleadas, y hay que aniquilarlos antes de que nos destruyan con sus armas. En los mejores juegos se recorren distintos niveles de dificultad.

var el despeñadero que nos bloquea el paso. Muchos de estos juegos emplean recursos gráficos.

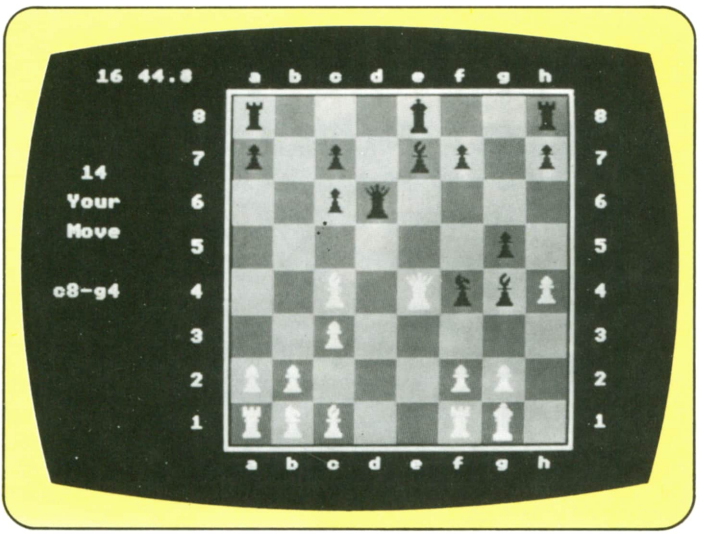
La fiebre de matar marcianos cobró verdadero impulso con la aparición de *Space Invaders* («invasores del espacio»), siniestras hileras de invasores que sueltan bombas que descienden lentamente por la pantalla, mientras que el jugador, con un cañón situado en la parte inferior de la misma,



La puerta del tiempo. En este juego de aventuras el jugador hace el papel de un explorador y goza de la satisfacción de matar algún que otro marciano por el camino. Estos juegos de aventuras son los herederos de algunos juegos de ordenador más antiguos.

trata de destruirlos antes de que ellos le destruyan a él. En máquinas instaladas en bares y salones recreativos *Space Invaders* se convirtió en una moda mundial y generó incontables imitadores, tanto en ese tipo de máquinas como en los ordenadores domésticos. El *Space Invaders* original ha sido superado hace ya tiempo por juegos más refinados, pero el principio básico permanece: unos seres extraños a los que hay que eliminar antes de que nos eliminen. Los mejores juegos cuentan con complejos sistemas de niveles de dificultad y asignación de puntos.

El tercer tipo de juegos, los tradicionales de mesa, ha encontrado aplicación en la investiga-



El ajedrez. El ajedrez es uno de los juegos más complejos y para los programadores resulta un auténtico desafío. Primero se pensó que los ordenadores nunca nos podrían derrotar, pero existen ya programas que se lo ponen difícil a los Grandes Maestros.

ción informática. Numerosos investigadores de la inteligencia artificial, con acceso a ordenadores potentísimos, se han interesado en particular por el ajedrez. Comprimir un programa de ajedrez en un micro, sin embargo, exige llegar a un compromiso con respecto a la potencia y al modo en que los ordenadores calculan realmente el movimiento que deben hacer. Pero con diestras técnicas de programación se han producido ya algunos programas de ajedrez muy potentes. Otros tipos de juegos de mesa, por desgracia, no resultan tan interesantes de jugar con un ordenador y no han tenido especial éxito, a excepción de algunas versiones de backgammon o las damas.

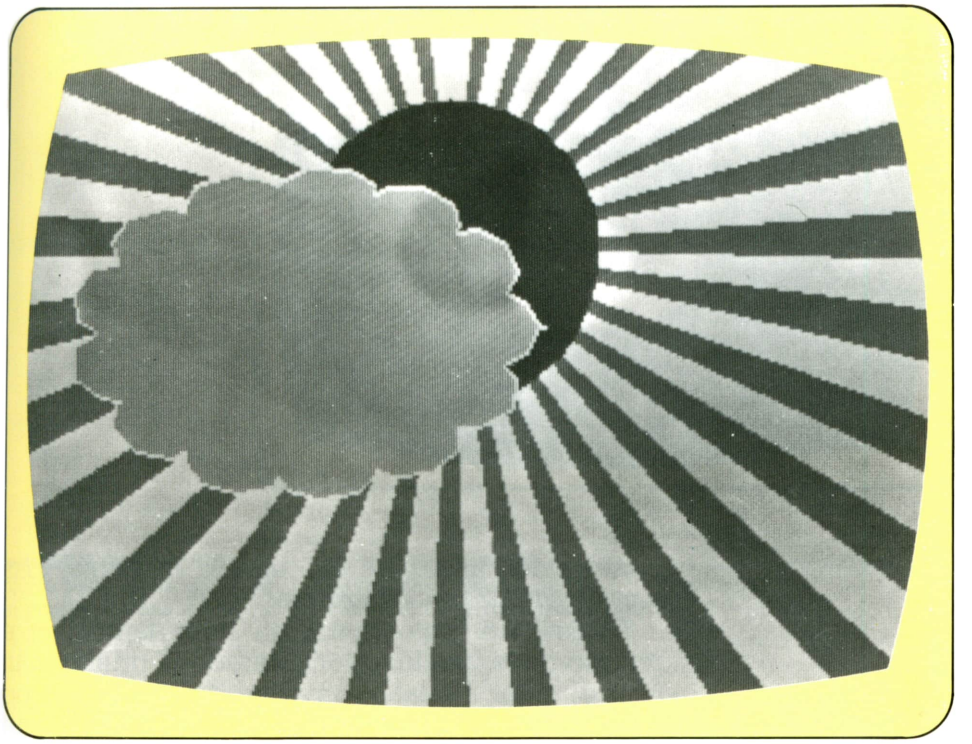
INFORMATICA GRAFICA

Durante algún tiempo se consideró que la informática gráfica constituía un tema de investigación interesante, pero sin demasiadas aplicaciones prácticas. La aparición y proliferación del micro hicieron cambiar de idea.

Los gráficos en el hogar

Hoy en día se consideran esenciales en un ordenador doméstico unos buenos recursos gráficos en color, y hay una lucha constante entre los fabricantes para que sean cada vez más completos. A algunos puede resultarles decepcionante que por el momento la calidad de los gráficos de los ordenadores domésticos no pueda competir con las de las máquinas *arcade*. Se debe a motivos técnicos relativos al empleo de televisores corrientes como pantalla gráfica. La conversión de la señal de vídeo del ordenador en señal de televisión exige el sacrificio de parte de la resolución de la señal original. El término resolución hace referencia al número de líneas de la pantalla y al número de puntos que componen cada línea: cuantos más puntos y líneas haya, mayor será el grado de resolución.

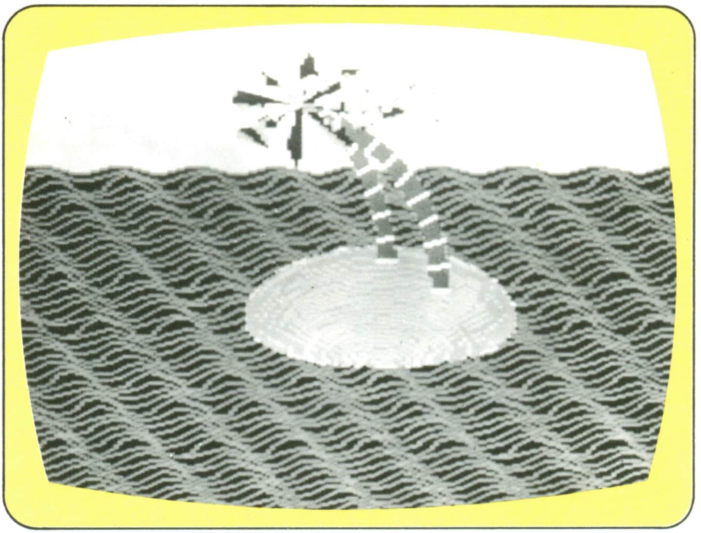
Algunos ordenadores domésticos tienen salidas de vídeo directas y modos gráficos especiales de resolución muy alta que pueden utilizarse con un monitor de vídeo.



Sistemas gráficos. Los avances en *software* y en *hardware*, junto con el descenso del precio de la memoria para implementar un sistema gráfico, han permitido que las máquinas domésticas se hayan visto enriquecidas con unas posibilidades gráficas de calidad notable. En ellas, la aplicación principal del color es añadir una nueva dimensión a los juegos, y está apareciendo continuamente un *software* cada vez más complejo. Pero, para que la calidad de las imágenes procedentes del ordenador iguale a las de la televisión comercial habrá que esperar a que se difunda el videodisco. También en las aplicaciones de gestión de empresas se combinan el color y los gráficos para presentar la información de modo más atractivo. Mediante los gráficos, la información resulta más fácil de asimilar, y añadir color supone contar con un canal más de información.

Los gráficos en la oficina

Cada día se usan más los gráficos en los ordenadores de gestión, sobre todo ahora que las máquinas de 16 bits gozan de más difusión. Por ejemplo, algunas máquinas ofrecen ahora una resolu-



1

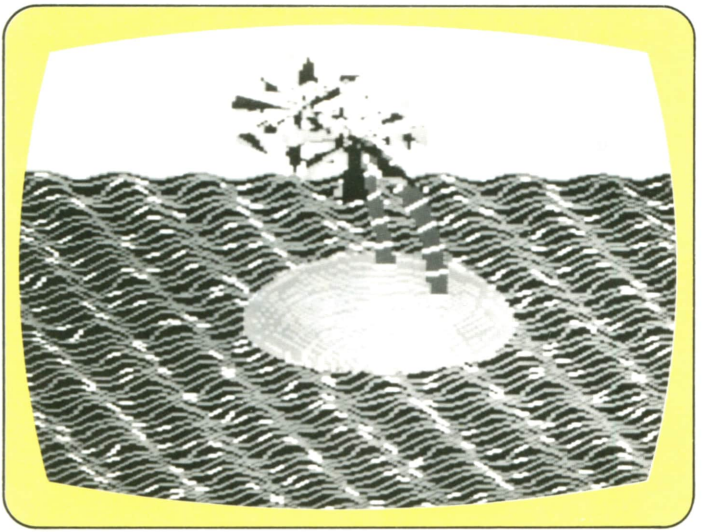
ción de 400 líneas de 800 puntos cada una, bastante elevada para un micro.

También el color gana en importancia en el mundo de la gestión. Considerado al principio un lujo, el color se está convirtiendo en un componente esencial de las máquinas de gestión a medida que todos se convencen de sus ventajas como vehículo suplementario de información, especialmente combinado con los gráficos. Una pantalla llena de cifras generadas por una hoja de cálculo, por ejemplo (véase Módulo 35), dice mucho más si la transformamos en un histograma o un gráfico. Muchas compañías de *software* están creando actualmente «paquetes» globales de tratamiento de la información que permiten presentar las cifras bajo muy diferentes modalidades gráficas.

Los gráficos y la memoria

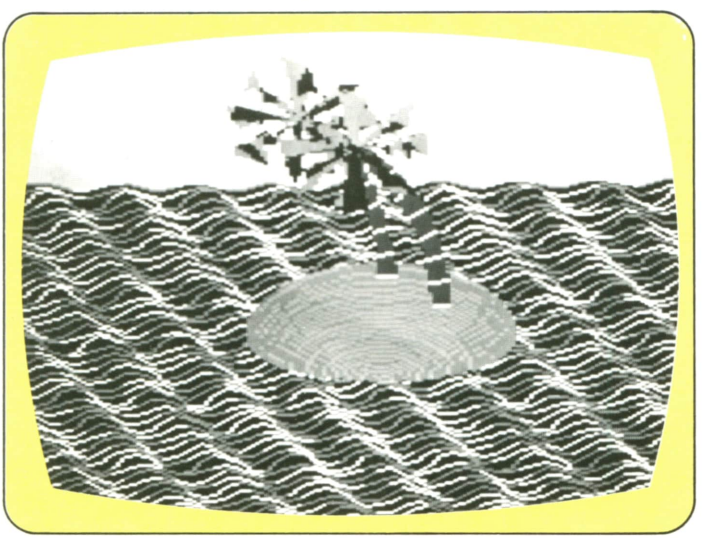
La creación de gráficos en color es relativamente sencilla y barata ahora que los *chips* de memoria tienen menor precio y mayores prestaciones. El método habitual consiste en reservar un «banco»

2



Secuencias gráficas. Algunos micros permiten la aparición en pantalla de una imagen al tiempo que se construye otra en una zona distinta de la memoria. La rápida unión entre una y otra crea sensación de movimiento.

3



de RAM, normalmente en la zona de memoria principal de la CPU en una máquina de 16 bits, o como unidad aparte «direccionada» a través de puertos de entrada-salida en una máquina de 8.

La verdadera labor de gobernar la pantalla la suele llevar a cabo un microprocesador llamado controlador de vídeo. En la RAM de vídeo, a cada bit le corresponde un punto de la pantalla. Para iluminar un punto, la CPU pone un 1 en la posición de RAM adecuada, y para apagarlo un 0. Este es el sistema llamado «de mapa de bits» (véase Módulo 20).

El *software* para gráficos está alcanzando una enorme complejidad, y muchos ordenadores domésticos cuentan ahora con potentes instrucciones gráficas incorporadas al intérprete de Basic, que permiten trazar y rellenar círculos, por ejemplo, o definir nuestro propio juego de caracteres si no nos gusta el que trae la máquina. El problema es que cada fabricante crea sus instrucciones, que no siempre coinciden. Esto, combinado con las diferentes gamas de color y resoluciones gráficas, hace que adaptar un programa en Basic de una máquina a otra resulte arduo.

Los gráficos en el futuro

Los micros, al menos por ahora, son incapaces de producir una imagen con la calidad de la televisión. Las investigaciones actuales en informática gráfica nos permiten vislumbrar lo que estas máquinas pueden dar de sí en el futuro. El proceso de describir matemáticamente un objeto para poder almacenar y manipular su «forma» es extremadamente complicado. Mostrar en una pantalla esta información exige unas operaciones matemáticas muy complejas, pero los resultados pueden ser asombrosos: una mano «humana», de suavidad irreal y piel perfecta, que se mueve lentamente, flexionando los dedos como si fuera de verdad. Otro ejemplo lo constituyen los cazas que vuelan en impecable formación bajo control del *joystick* del ordenador.

PROGRAMACION EN ENSAMBLADOR

En el Módulo 27 vimos cómo se programaba un ordenador en Basic. El Basic pertenece a la familia de lenguajes de programación llamados «de alto nivel», caracterizados porque sus instrucciones se dan mediante órdenes basadas en el inglés. Una de las ventajas de estos lenguajes de alto nivel es que un programa escrito y comprobado en una máquina se puede ejecutar sin problemas en otra marca totalmente distinta, que cuente también con un intérprete o compilador de ese lenguaje. Un inconveniente es que quizás el programa resultante no sea todo lo eficiente que pudiera. Los programas escritos en un lenguaje interpretado suelen ser lentos, y los escritos en uno compilado, aunque más rápidos, ocupan mucha memoria.

El transporte de programas

En la mayoría de los ordenadores se pueden escribir programas utilizando directamente las instrucciones binarias de la máquina, y suelen ser extremadamente rápidos y compactos. Pero, aparte de ser tedioso y consumir mucho tiempo, el principal inconveniente de estos programas es que no serán transportables; es decir, que a menos que los hayamos escrito para alguno de los sistemas operativos más populares, tales como el CP/M (véase Módulo 33), podemos tener graves proble-

mas a la hora de tratar de ejcutarlos en un ordenador de otra marca. Pese a ello, hay algunas aplicaciones en que resulta esencial programar a bajo nivel, sobre todo en rutinas cuya velocidad es crítica o para crear *software* que vaya a residir en ROM, lo que significa forzosamente disponer de una cantidad de memoria limitada.

¿Qué es el lenguaje ensamblador?

Es virtualmente imposible escribir un programa de más de unas cuantas docenas de octetos en código de máquina rápidamente y sin equivocarse. Escribir en binario, mejor ni pensarlo; y ni siquiera en hexadecimal resulta mínimamente satisfactorio. Este es el problema que viene a resolver el lenguaje ensamblador, un lenguaje abreviado en el que en lugar de las instrucciones binarias se utilizan mnemónicos. Los programas en ensamblador se escriben mediante un editor o procesador de textos, se guardan en cinta o disco como ficheros de texto normales, y luego se indica a un programa también llamado ensamblador que lea el fichero y lo convierta en código de máquina binario directamente ejecutable por el ordenador.

Antes de empezar a programar en ensamblador es preciso estudiar el procesador que se va a usar y, a menudo, ciertos detalles del *hardware,* tales como las direcciones de los puertos de entrada-salida. La mayoría de los procesadores tiene su juego de instrucciones particular e incompatible con los demás, juego que hay que aprender para poder programar un procesador concreto. En este módulo tomaremos como ejemplo el Z80 de Zilog. En un ordenador cuyo procesador sea diferente, el código no funcionará, pero los principios generales seguirán siendo válidos y adaptarlo no debe ser un problema grave.

El procesador Z80 por dentro

Estudiar el procesador antes de intentar escribir un programa en ensamblador significa examinar su «arquitectura» interna. El Z80 tiene varios «registros» o posiciones internas de memoria. Son 22

en total, estando los más importantes duplicados en un bloque de registros secundario, y contándose con una instrucción especial que permite cambios a voluntad entre los dos bloques (véase Módulo 17). Nada añadiremos sobre los registros secundarios, pues todo lo dicho para el bloque principal también será aplicable a ellos.

Cada registro del bloque principal tiene un nombre y su capacidad es de un octeto. Al registro fundamental se le llama «acumulador», y es ahí donde tienen lugar las principales operaciones, pues normalmente es preciso llevar un octeto al acumulador para someterlo a alguna operación de tipo aritmético o lógico.

Relacionado con el acumulador está el registro «de banderas». Este, a diferencia de otros registros, es una colección de celdas de memoria de un bit, dos de los cuales no se usan. Las más importantes son la «bandera de arrastre» y la «bandera de cero», que indican, respectivamente, si en la operación que acaba de realizarse en el acumulador ha habido arrastre, o si su contenido tras la operación es cero. Estas banderas permiten, en conjunción con diversas instrucciones, determinar el camino a seguir. Por ejemplo, si al restar un número del acumulador sale cero, bifurcar a otra parte del programa, y si no, seguir con la siguiente instrucción.

Los restantes registros del bloque principal son registros polivalentes, y se llaman B, C, D, E, H y L. Aparte de como registros de 8 bits, también pueden usarse por parejas para operar con valores de 16 bits. En este caso, los registros reciben los nombres de BC, DE y HL. Ejemplo de esta práctica podría ser cargar en HL una dirección de memoria para coger el octeto que se encuentra en esa posición y cargarlo en el acumulador.

De los demás registros, el único que nos interesa ahora es el «puntero de pila» o SP *(stack pointer)*. La pila es una zona de memoria reservada para ciertos usos, por ejemplo, en las llamadas a subrutinas. Vimos en el Módulo 27 que si una operación había que repetirla muchas veces, lo mejor

era escribirla aparte del resto del programa y recurrir a ella cuantas veces fuera preciso mediante la instrucción del Basic GOSUB. En la programación en ensamblador existe un recurso parecido, sólo que en lugar de GOSUB se emplea CALL. Cuando se ejecuta una instrucción CALL, el procesador guarda en la pila la dirección en memoria de la instrucción siguiente a la CALL. La pila funciona como ese mecanismo de muelle para apilar platos que hay en algunos restaurantes: si añadimos otro plato al montón, los demás bajan un lugar, y si quitamos uno, suben. Este principio lo remeda en el ordenador el puntero de pila. Contiene la dirección de una posición de memoria vacía donde se puede almacenar temporalmente un octeto. Si añadimos un octeto a la pila (instrucción PUSH) el puntero se reajusta automáticamente, pasando a señalar la siguiente posición de memoria disponible, y se copia el valor en esa posición. Cuando se llama a una subrutina mediante CALL, el procesador pone la dirección de la instrucción que viene detrás de la CALL en la pila y luego pasa a ejecutar la subrutina. Cuando llega a la instrucción RET, última de la subrutina, recupera la dirección de vuelta, sacándola de la pila (instrucción POP) y prosigue la ejecución del programa a partir de ella. En muchos programas, lo primero que hay que hacer es «inicializar» el puntero de pila (cargándole una posición de memoria que será la que se utilice como pila). En la mayor parte de los procesadores, el Z80 entre ellos, se pone en el puntero de pila una posición próxima al extremo superior de la memoria RAM disponible y la pila crece hacia abajo.

La programación en ensamblador en la práctica

Vamos a examinar algún ejemplo de programa en el ensamblador del Z80. A la derecha podemos ver un pequeño listado en el formato habitual de los programas en ensamblador. Las columnas empleadas son, de izquierda a derecha, etiqueta, código de operación, operandos y comentarios.

LISTADO EN ENSAMBLADOR

```
        ORG     100H            ;Dirección de comienzo
                                ;del programa
        CR      EQU   0DH       ;Retorno de carro ASCII
;
;Sección de entrada del programa
;
START:  LD      HL,BUFFER       ;Aquí empezamos cargando HL
                                ;con la dirección del área de
                                ;memoria de los caracteres que
                                ;que se van a almacenar.
LABEL1: CALL    GETIT           ;Llamada a la subrutina que toma
                                ;el carácter tecleado
        LD      (HL),A          ;Almacena el carácter
        INC     HL              ;Incrementa HL al punto de la
                                ;nueva posición de memoria
        CP      CR              ;¿Es el carácter tecleado RETURN?
        JR      NZ,LABEL1       ;En caso contrario, retrocede a
                                ;a tomar el nuevo carácter
;
;Esta es la sección de salida
;
        LD      HL,BUFFER       ;Restaura HL al punto del primer
                                ;carácter
LABEL2: LD      A,(HL)          ;Toma el primer carácter y lo
                                ;pone en el acumulador
        CALL    PUTIT           ;Llama a la subrutina que visualiza
                                ;el carácter
        INC     HL              ;Incrementa HL al punto del nuevo
                                ;carácter
        CP      CR              ;¿Es este carácter RETURN?
        JR      NZ,LABEL2       ;En caso contrario, retrocede y toma
                                ;el nuevo carácter de memoria
;
;Aquí termina el programa, volviendo en este caso al CP/M
;
        JP      0
;
;A continuación está la subrutina que toma el carácter tecleado
;Esta subrutina está escrita para el sistema CP/M, si su sistema
;es distinto necesitará reescribirla
;
GETIT:  PUSH    BC              ;Esta subrutina usa el registro C
                                ;y es una buena práctica, para poner
                                ;(PUSH)cualquier registro usado en
                                ;la subrutina en el "stack" (memoria
                                ;intermedia), para guardar su contenido
                                ;en caso de que sea necesario para el
                                ;programa principal
        LD      C,1             ;Función CP/M para tomar un carácter
                                ;desde el teclado
        CALL    5               ;Llamada a la subrutina CP/M para leer
                                ;el teclado- el carácter es devuelto en A
        POP     BC              ;Restaura el contenido de BC desde el
                                ; "stack"
        RET                     ;Vuelve al programa principal
;
;Esta subrutina visualiza en la pantalla  el carácter del acumulador
;también necesita ser reescrita  para sistemas no CP/M
;
PUTIT:  PUSH    BC              ;Salva estos registros porque lo usa la
        PUSH    DE              ;subrutina
        LD      C,2             ;Función CP/M que visualiza el carácter en
                                ;la pantalla.
        LD      E,A             ;El CP/M requiere que el carácter sea puesto
                                ;en el registro E, al que lo transferimos
                                ;desde A
        CALL    5               ;Llama a la rutina CP/M para visulizar un
                                ;carácter
        POP     DE              ;Ahora restaura estos registros, pero
        POP     BC              ;recuerde que el "stack" opera en modo
                                ;último en entrar-primero en salir (LIFO)
                                ;debemos emplear el POP en sentido inverso
                                ;para que tome su contenido correcto
        RET                     ;Vuelve al programa principal
;
;Aquí definimos el área que se va a usar como zona de almacenamiento
;llamada BUFFER
;
BUFFER: DEFS    80              ;DEFS es una instrucción del Assembler, no
                                ;del ordenador, reserva un área de memoria
                                ;(80 bits en este caso), en la que BUFFER
                                ;es la dirección del primer bit.
;
;¡Y aquí termina el programa!
;
```

Una «etiqueta» es un nombre simbólico que normalmente representa una dirección en memoria. En este programa tenemos cuatro casos: PUTIT y GETIT, por ejemplo, son los nombres de las dos subrutinas. En lugar de trabajar con las direcciones exactas que tienen en memoria las dos subrutinas, en el programa principal las llamamos con unos nombres, y luego, al escribir las subrutinas, les ponemos delante una etiqueta con esos nombres. El ensamblador calculará más tarde las direcciones verdaderas y las pondrá en la instrucción CALL.

Un «código de operación» es, sin más, el nombre que recibe una instrucción en la jerga del ensamblador.

Un «operando» le dice al ordenador a qué se refiere el código de operación. Por ejemplo, en la primera línea del programa propiamente dicho, la que lleva la etiqueta START, el código de operación es LD (mnemónico de *load,* cargar) y el operando consta de un nombre de registro (el HL) y de una dirección de memoria (BUFFER). En otras palabras, lo que le estamos diciendo a la CPU es que cargue en la pareja de registros HL la dirección de la etiqueta BUFFER, definida en la última línea del listado como una zona de 80 octetos (BUFFER: DEFS 80). No todos los códigos de operación necesitan operandos: la instrucción RET que va al final de las subrutinas, por ejemplo, no lleva ninguno.

Un «comentario» se inserta en el programa por comodidad del programador, y funciona igual que la sentencia REM del Basic. Aquí se pone punto y coma y el ensamblador ignora todo lo que vaya detrás de esa línea. En realidad, los comentarios pueden empezar en cualquier columna. Esto puede aprovecharse para descomponer el programa en secciones fáciles de identificar.

Antes de examinar en detalle lo que hace, conviene aclarar un par de cosas sobre este programa. Como la programación en ensamblador es muy dependiente de la máquina, el programa se ha escrito para el sistema operativo CP/M (véase

Módulo 33). En una máquina que no soporte CP/M habrá que modificarlo. Otra cosa: las dos primeras líneas del programa son especiales. No forman parte del programa: son instrucciones para el propio programa ensamblador. La primera (CR EQU,ODH) asigna el valor OD hexa (el código ASCII del retorno de carro) al nombre «CR». Esto significa que se puede poner CR en el programa siempre que se quiera uno referir a dicho código, con lo que resulta más fácil entender el listado. Y la sentencia ORG 100H dice al ensamblador la dirección de memoria en que empieza el programa. Como se trata de un programa CP/M ha de empezar necesariamente en 100 hexa, pero en otros ordenadores puede ser diferente.

Análisis del programa

El programa en sí es muy sencillo, pero sirve para aclarar varias cuestiones importantes. Lo que hace es tomar una serie de caracteres tecleados por el usuario y almacenarlos en una zona de memoria llamada BUFFER. El programa sigue admitiendo caracteres hasta que el usuario teclea un retorno de carro, y a continuación escribe en la pantalla todos los caracteres leídos.

Las partes del programa que habrá que alterar para adaptarlo a otra máquina son las dos subrutinas (GETIT y PUTIT) que sirven, respectivamente, para recoger un carácter tecleado y para escribir un carácter en pantalla. GETIT espera a que tecleemos un carácter y cuando lo recibe lo mete en el acumulador (esto se hace automáticamente mediante una llamada a CP/M); PUTIT supone que el acumulador contiene un carácter y lo escribe en pantalla.

El programa comienza cargando en la pareja de registros HL la dirección de la zona de memoria donde se van a almacenar los caracteres, que es BUFFER. Luego llama a la subrutina GETIT, que devuelve en el acumulador, o registro A, el carácter recibido. A continuación se utiliza la dirección contenida en HL como «puntero» (se indica po-

niendo HL entre paréntesis) y se transfiere el carácter que hay en el acumulador a esa posición de memoria. Después se «incrementa» la dirección contenida en HL, es decir, se le suma 1, de manera que señale a la siguiente posición desocupada de BUFFER. Miramos entonces, mediante la instrucción CP CR, a ver si el carácter tecleado ha sido un retorno de carro. Esa instrucción compara el contenido del acumulador con el valor de «CR», que definimos como ODH (el código ASCII del retorno de carro) al principio del programa. Si son iguales, se pone a 1 la bandera de cero. La siguiente instrucción examina precisamente la bandera de cero: JR NZ, LABEL1 significa que si el resultado no es cero hay que volver a la dirección especificada mediante la etiqueta LABEL1 y continuar a partir de ahí. Por tanto, si el carácter tecleado no ha sido retorno de carro, volvemos a llamar a GETIT y a esperar a que llegue otro carácter del teclado.

Si, por el contrario, el carácter era retorno de carro, el programa pasa a la sección encargada de escribir en pantalla. En ella se empieza por volver a cargar en HL la dirección de BUFFER, y a continuación se vuelve a emplear HL como puntero, pero esta vez se carga el contenido de esa dirección en el acumulador con LD A, (HL). Luego se llama a PUTIT para escribir el carácter en pantalla y se incrementa HL para que señale al siguiente carácter. Se examina el contenido del acumulador para ver si es retorno de carro, lo que significaría el final de la cadena de caracteres. Si no lo es, se vuelve atrás para extraer de la memoria el siguiente carácter.

El programa termina con una instrucción que devuelve a CP/M el control del ordenador, aunque en otra máquina no basada en CP/M seguramente habrá que poner una dirección distinta. Las instrucciones de bifurcación JP y JR no son iguales. JP debe ir seguida de una dirección de 16 bits (cero en este caso, pero podía ser una etiqueta), y por tanto puede entregar el control del progra-

ma a cualquier posición de la memoria del ordenador. JR, por su parte, significa «Bifurcación relativa» *(Jump Relative)* y utiliza un operando de 8 bits. El ordenador «conoce» la dirección de memoria de la instrucción que está ejecutando en ese momento (tiene que conocerla para poder extraer la instrucción de memoria) y suma o resta el operando a esta dirección para obtener la posición de la instrucción a la que debe bifurcar. Por ello su alcance es más limitado (de 126 octetos antes de la instrucción hasta 129 detrás en el caso del Z80). Pero en muchos bucles sobra con ese intervalo, y usando JR en lugar de JP se ahorra un octeto de memoria y la ejecución es más rápida.

Máquinas sin ensamblador

Existen ordenadores domésticos que no disponen de ensamblador. Si este es el caso, tendremos que ensamblar a mano nuestro programa, traduciendo a hexa cada código de operación. Hecho esto, y resueltas las bifurcaciones relativas, lo normal es que tengamos que pasarlo a decimal (existen calculadoras especiales para esto, y también podemos escribir un sencillo programa en Basic), incluir los números resultantes en un programa en Basic mediante sentencias DATA, y meterlos en una zona de memoria apropiada mediante la instrucción POKE. Suele haber en los Basic una instrucción como USER(nnn) o algo parecido, en la que nnn es la dirección decimal del comienzo de la subrutina en código de máquina a la que se quiere ceder el control.

OTROS LENGUAJES DE PROGRAMACION

Los términos que los hombres entendemos y que nos sirven para plantear los problemas no le resultan comprensibles al ordenador, y el lenguaje de los ordenadores es difícil si no imposible de digerir para una persona salvo en reducidísimos niveles.

¿Por qué diferentes lenguajes de programación?

El programador se ciñe a emplear un vocabulario limitado, especial, que el mismo ordenador puede traducir a los términos que le resultan inteligibles. Analizando un problema y expresando la solución en este idioma intermedio, el programador consigue ponerlo al alcance del ordenador aun cuando el planteamiento (o programa) no deje de estar expresado en una forma comprensible por cualquier persona versada en ese lenguaje de programación.

El proceso de diseñar un lenguaje de programación, por tanto, se reduce a analizar el tipo de cosas que normalmente se piden a un ordenador y escribir unas rutinas generales que las lleven a cabo. Viendo, por ejemplo, que muchos programas han de mostrar información por pantalla, puede incluirse en el lenguaje una rutina llamada

PRINT que se encargue de ello. Ulteriores puntualizaciones potenciarán su versatilidad.

Como con los ordenadores se puede hacer muchas cosas distintas, si se quiere satisfacer las necesidades de todo el mundo habrá que dotarlos de posibilidades incontables. Y a veces se ha pretendido crear un lenguaje único y general, aunque fuera difícil de aprender para el programador y complicado de traducir a la forma inteligible por la máquina. Pero no es fácil desarrollar un lenguaje de esta magnitud para las grandes máquinas, ni posible para los microordenadores.

Por este motivo existen diversos lenguajes de programación. Ninguno de ellos aspira a ser el lenguaje perfecto, pero cada uno se diseñó pensando en cierto tipo de trabajo. Con el Basic, por ejemplo, se quiso crear un lenguaje pedagógico, fácil de aprender, que permitiera al no especialista utilizar un ordenador sin tener que dedicar antes mucho tiempo a aprender la nueva técnica. Con el Pascal, se quiso crear un lenguaje polivalente, rígidamente estructurado, con la idea de enseñar buenos hábitos de programación, obligando al estudiante a atenerse estrictamente a las reglas. El Cobol, concebido en un principio como un lenguaje para profanos, es hoy fundamentalmente una herramienta para programas de gestión.

En este módulo examinaremos algunos lenguajes de programación populares, disponibles todos ellos en algún microordenador, cada uno de los cuales tiene su campo de aplicación particular. Por ejemplo, nunca nos pondríamos a escribir un procesador de textos en Cobol si hay a mano un compilador de C; el control de un equipo de laboratorio y la adquisición de datos se harían mejor en Fortran que en un Basic interpretado, sobre todo si es importante la velocidad de ejecución.

Un hecho curioso de la programación es que cada lenguaje tiene sus fervientes partidarios. También los métodos de programación suscitan debates no menos apasionados: una de las escuelas de pensamiento propugna el método estructu-

rado, fundado en una cuidadosa descomposición del programa en secciones autónomas claramente separadas, de manera que el flujo discurra ordenadamente y sin interrupciones dentro de cada sección. Este método es muy razonable pero, por desgracia, no siempre funciona en la práctica.

El propósito que preside este módulo es dar al lector una idea de las características de algunos de los lenguajes, poniendo de manifiesto sus ventajas e inconvenientes y comentando algunos de sus rasgos peculiares; nunca tratar de enseñarle a programar en todos ellos.

C

El C es probablemente el más versátil de los lenguajes de programación disponibles en microordenadores. Deriva de un lenguaje de programación británico llamado BCPL que adoptaron los Laboratorios Bell, en Estados Unidos, como base de un lenguaje llamado B, que, a su vez, mejorado y modificado, acabó por cristalizar en el lenguaje hoy llamado C.

El C se concibió en un principio para el sistema operativo Unix, aunque luego se haya implementado en muchos otros, tanto en macro como en miniordenadores, así como en algunos micros. Posiblemente sea el lenguaje más transportable. Se puede escribir un programa en C para cualquier ordenador y llevar el código fuente (el fichero de texto que contiene las instrucciones que luego traduce el compilador a código de máquina) a otro completamente diferente tal como está con casi un cien por cien de probabilidades de que funcione sin ninguna modificación. Las únicas excepciones son algunas secciones muy dependientes de la máquina, tales como las que se ocupan de gestionar la pantalla.

A primera vista, un programa C parece incomprensible. Aunque contiene algunas instrucciones en inglés de significado bastante obvio, muchas expresiones están abreviadas. Sin embargo, en cuanto empezamos a aprender el lenguaje las co-

sas se van aclarando, y enseguida se llega a tener en alta estima la concisión del C. Se pueden escribir programas en lo que viene a ser una notación taquigráfica, lo que es de agradecer sobre todo si se procede del Cobol o del Pascal.

El C ocupa una posición especial en la jerarquía de los lenguajes. Es un lenguaje de alto nivel,pero, al mismo tiempo, nos permite descender al nivel del bit, como si estuviéramos trabajando en ensamblador (véase Módulo 31). Es por tanto un lenguaje ideal para escribir *software* de sistemas (véase Módulo 35). La mayor parte del sistema operativo Unix está escrito en C y la firma Digital Research (introductora del CP/M) utiliza ahora C en todos sus sistemas operativos, compiladores e intérpretes.

El C es semejante al Pascal en ciertos aspectos. Ambos son lenguajes estructurados que fomentan el uso de la programación modular, que facilita tanto la comprensión de los listados. Sin embargo, el C es menos autoritario que el Pascal y no impone reglas tan estrictas sobre la organización del programa. Su sintaxis es asimismo menos severa y prolija, lo que es sin duda una de las causas de su popularidad, aunque sus principales atractivos residan en su transportabilidad y versatilidad.

Un programa en C consta de un módulo principal y diversas funciones, que equivalen aproximadamente a las subrutinas del Basic (véase Módulo 27). Aparte del requisito de identificar con el nombre «main» el módulo principal, está ausente el rigor del Pascal en el tema de la posición en el programa de dicho módulo. En Pascal, toda función hay que definirla antes de usarla, por lo que un listado de Pascal comienza con la función de «nivel más bajo» y concluye con el programa principal. En C, por el contrario, podemos colocarlas en el orden que queramos.

Todas las variables que se usan en un módulo han de estar definidas al comienzo del módulo, y es imprescindible declarar su tipo. Los tipos soportados son: entero, carácter y coma flotante y,

en la mayoría de las máquinas, se soportan también los enteros largos y cortos y diversos tipos de coma flotante. El C posee todos los operadores aritméticos habituales: +, –, * (multiplicación) y / (división), aparte del «operador módulo» (da el resto al efectuar una división), pero no las funciones «trascendentes» que otros muchos lenguajes incorporan, tales como el seno y el logaritmo. Cuenta, eso sí, con varios operadores lógicos, que permiten trabajar con números enteros o a nivel de bit: AND, OR y XOR, por ejemplo.

Uno de los rasgos más apreciados del C es la economía de expresión que permite. Consideremos, por ejemplo, lo siguiente (los símbolos «/*...*/» se usan para encerrar comentarios, y equivalen a la sentencia REM del Basic):

```
int  x;    /* La variable x es declarada
como entero ... */
x = 1;     /* ... y le damos valor 1 */
x = x + 10 ; /* x la incrementamos en 10.*/
```

Esto podríamos escribirlo:

```
int x = 1 ; /* El valor de x es definido
y la variable declarada */
x += 10 ; /* Forma rápida de decir
x = x + 10 */
```

El equivalente del IF...THEN...ELSE del Basic es la sentencia «if», que se usa de esta manera:

```
if ( x = 20 )
   x = 1;
else
   x = 0;
```

En C se pueden agrupar varias líneas del programa encerrándolas entre los símbolos { y }, con lo que serán consideradas como una sola sentencia. Todas las sentencias de un módulo del programa deben ir encerradas entre llaves. También se pueden usar en una sentencia IF:

```
if ( x = 20 )
   {
   x = 1;
   y = 2;
   }
```

Hay tres formas de controlar bucles en C:

```
for ( x = 1; x == 10; ++x )
   y += 6;
```

Esto es parecido al bucle FOR...NEXT del Basic. Con x=1 se inicializa x; x==10 comprueba (mediante el uso del operador ==) si x ha alcanzado el valor 10. Si es así, cesa la ejecución del bucle; ++x suma 1 a x. Esta sentencia equivale a FOR x=1 TO 10 en Basic, y la comparación para ver si hay que llevar a cabo una iteración más, tiene lugar al principio del bucle.

```
x = 1;
while ( x != 10 )
   {
   y += 6;
   ++x;
   }
```

Este fragmento de programa es otra forma de realizar el mismo bucle, comprobando la condición también al principio. Nótese la comparación !=, que significa «no es igual». En este caso, la construcción «for» es más clara, pero en ocasiones es preferible el bucle «while». Y otras veces se desea hacer la comprobación al final del bucle:

```
x = 1;
do {
    y += 6;
    } while ( ++x != 10 );
```

Aquí se comprueba después de cada iteración del bucle, incrementando la x mediante ++x.

Los valores se pasan a las funciones escribiéndolos dentro de las llaves que van detrás del nombre de la función; si la función no precisa de ninguno, las llaves deben estar también presentes, aunque no haya nada dentro. El C permite usar la llamada a la función como si fuera una variable, y se puede poner en cualquier sitio que admita una variable. Es muy corriente en un listado de C encontrar una sentencia condicional en la que una o más de las condiciones son llamadas a funciones o incluso llamadas a función «anidadas», es decir, que se pasa un valor a una función que es a su vez una llamada a otra función.

Una particularidad del C es que el lenguaje propiamente dicho no tiene ningún recurso de entrada-salida. Esto parece una limitación tremenda. ¿De qué sirve un programa que no puede recibir ni suministrar datos de ningún tipo? Existen, desde luego, recursos de E/S, pero revisten la forma de llamadas a funciones, no de palabras pertenecientes al lenguaje. Cada compilador de C lleva su librería normalizada de funciones de E/S, y en ello reside una de las claves de la transportabilidad del C. Al agrupar todos los recursos de E/S y normalizar su nombre y su sintaxis, el lenguaje

libera al programador de tener que preocuparse de los detalles de la E/S de la máquina o del sistema operativo con que está trabajando. El C soporta los habituales teclado, pantalla e impresora, más las funciones necesarias para la gestión de ficheros, incluidos los de acceso aleatorio. Una de las ventajas del C es que la E/S se puede reconducir desde dentro de un programa o bien desde el sistema operativo cuando se ejecuta un programa, con lo que puede conseguirse fácilmente que su salida (que normalmente se dirige a la pantalla) vaya a parar a una impresora o a un fichero.

Posiblemente, las partes del C que más desconcierto siembran son las variables dimensionadas y los punteros. En C hay variables dimensionadas numéricas como en cualquier otro lenguaje, pero además las cadenas se consideran variables dimensionadas cuyos elementos son los caracteres. No tiene previsto manipular cadenas enteras, siendo este uno de sus inconvenientes. A diferencia de otros lenguajes, el C nos permite acceder a un elemento de una variable dimensionada no sólo especificando su posición en dicha variable mediante un número o una variable, sino también especificando su dirección verdadera en memoria mediante un «puntero». No disponemos aquí de espacio para dar una explicación cumplida de los punteros, pero sus efectos generales son tres: crear listados ilegibles, llevarnos al caos cuando estamos aprendiendo a usarlos, y generar un código más rápido, compacto y eficaz una vez que se llega a dominarlos.

El C es un lenguaje muy útil, pero no se adapta bien a cierto tipo de aplicaciones, como gestión, ciencias y matemáticas. El C es incomparable para escribir sistemas operativos y aplicaciones con versatilidad próxima a la del lenguaje ensamblador.

Forth

El Forth es un lenguaje de programación de lo más particular. Parece difícil de entender, apren-

der y utilizar. Sin embargo, familiarizados con él es extraordinariamente potente, rapidísimo y, por su peculiar facultad de aceptar extensiones de sí mismo, quizás el lenguaje de programación más flexible.

El Forth fue creado por el estadounidense Charles Moore, descontento de los lenguajes de programación convencionales. Convencido de que éstos no le daban la potencia, sencillez y flexibilidad que necesitaba, resolvió diseñar uno nuevo, usando un ordenador IBM 1130. Por aquellos años, el 1130 contaba entre los ordenadores de la «tercera generación», así que, como el nuevo lenguaje de Moore parecía tan potente, decidió llamarlo *Fourth* («cuarto»). Por desgracia, el 1130 sólo admitía identificadores de cinco caracteres, por lo que se vio obligado a dejarlo en «Forth».

El Forth es algo más que un lenguaje: una combinación de lenguaje y sistema operativo. Su núcleo es el «diccionario», una colección de órdenes y palabras. Uno de los aspectos que dotan al Forth de potencia y flexibilidad es que el usuario puede incorporar al diccionario sus propias palabras, lo que equivale a incorporarlas al lenguaje. Sigamos con un ejemplo el proceso de incorporar una nueva palabra al Forth. La palabra del Forth «n EMIT», en la que n es el valor decimal de un carácter ASCII, provoca la aparición en pantalla de ese carácter. Si tecleamos «42 EMIT», saldrá por pantalla un asterisco, ya que 42 es el código ASCII en decimal del asterisco (véase Módulo 13) Podemos ahora definir una palabra nueva que escriba por pantalla determinado número de asteriscos cuando tecleemos simplemente «n STARS» de la siguiente manera:

```
: STARS  0 DO 42 EMIT  LOOP  CR ;
```

Los dos puntos le indican al Forth que la siguiente palabra es nueva, que la vamos a definir. Se observa el "42 EMIT" en el seno del bucle que le hace

repetirse el número de veces especificado cuando se teclea STARS. El "CR" es para que el Forth lance un retorno de carro una vez escritos los asteriscos, y el punto y coma del final sirve para dar por terminada la definición. Después de proporcionarle esta definición podemos teclear simplemente "10 STARS" y el Forth nos escribirá una fila de diez asteriscos. Las palabras nuevas pueden usarse luego para definir otras. Veamos este ejemplo:

```
: STAR    42 EMIT ;
: STARS    0 DO  STAR  LOOP  CR ;
: LINE    10 STARS ;
: GAP    STAR  8 SPACES  STAR  CR ;
: BOX    CR  LINE  GAP  GAP  LINE  CR ;
```

Las dos primeras líneas ya las hemos visto. En la tercera línea definimos «LINE», que escribe una línea de diez asteriscos. «GAP» escribe un asterisco, y luego ocho espacios en blanco, aprovechando la palabra «SPACES», seguidos de otro asterisco y un retorno de carro. Por último definimos «BOX» de manera que realice un retorno de carro, luego una línea de diez asteriscos, luego dos líneas, cada una de ellas con un asterisco, ocho blancos y un asterisco, luego otra línea de asteriscos, y otro retorno de carro. Si después tecleamos BOX, obtendremos la siguiente imagen:

```
**********
*        *
*        *
**********
```

Todas las «implementaciones» del Forth cuentan con un Editor que permite crear programas en Forth. Una vez escrito el programa, se puede verificar interactivamente desde Forth y, comproba-

do que funciona, compilarlo e incorporarlo el diccionario. Con este sistema puede llegarse a construir un programa muy complejo, pero que inmediatamente pasa a formar parte del Forth y puede activarse sin más que teclear una palabra.

Como todas las nuevas palabras se compilan en el momento de incluirlas en el Forth (se incorporen o no al diccionario), la ejecución es extremadamente rápida. Una palabra de Forth compilado no se ejecutará tan deprisa como el código equivalente escrito en ensamblador, pero el Forth también permite incorporar partes en código ensamblador a nuestras nuevas definiciones para los casos en que la velocidad sea importante.

El Forth probablemente no sea el lenguaje ideal para aprender el primero, pues para usarlo eficazmente es preciso entender bien determinados conceptos informáticos. Uno de los principales es el de pila (véase Módulo 31), que en el Forth desempeña un papel mucho más importante que en la mayoría de los lenguajes. En un ordenador, la pila es una zona de memoria especial y reservada. Se pueden meter *(push)* y sacar *(pop)* datos en la pila según el principio de «último en llegar, primero en salir». Puede decirse que el Forth usa la pila en todas las operaciones en las que hay que hacer algo con uno o más datos. Tomemos, por ejemplo, la sencilla operación de sumar dos números. En Basic se puede hacer así:

```
PRINT 3+4
```

y obtener la respuesta, 7. En Forth, sin embargo, las operaciones matemáticas se expresan mediante una notación diferente, llamada polaca inversa o de sufijo.

La operación anterior se escribiría en Forth:

```
3 4 +
```

Lo que ocurre es que el Forth va tomando los números, el 3 y el 4, y los mete en la pila. Luego encuentra el operador +, que le indica que ha de sumar los dos números que están en lo alto de la pila, así que los saca, los suma, y escribe el resultado. El mismo principio actúa en la definición de BOX que veíamos antes. Si tecleamos "42 EMIT" primero el número 42 pasa a la pila, y luego "EMIT" saca el número que está en lo alto de la pila y escribe el carácter resultante de considerarlo un código ASCII. Análogamente, SPACES saca el primer número de la pila y lo usa como contador en un bucle que escribe espacios en blanco en la pantalla.

Trabajar en esta notación de sufijo resulta al principio engorroso (salvo que se tenga una calculadora Hewlett-Packard, que también la usa), pero es mucho más fácil si dedicamos cierto tiempo a entender bien el funcionamiento de la pila y lo que hace el Forth cuando le solicitamos operación en la que intervenga dicha pila.

También sirve en Forth la pila (y en muchos otros lenguajes) para pasar valores de una operación a otra. A veces, una operación mete valores en la pila en cierto orden, distinto del que otra operación requiere. En previsión de esto existen en el Forth unas palabras que permiten manipular el contenido de la pila. Se puede copiar cualquier cosa que haya en la pila sin modificar su contenido, o sacar algo que haya más abajo, no exactamente lo último que entró. También se puede reordenar su contenido. Para efectuar estas operaciones hace falta práctica, pues los errores suelen ocasionar un caos increíble.

El Forth, sin embargo, tiene muchos partidarios entre los aficionados y los profesionales, y en él se han escrito algunas aplicaciones interesantes.

Pascal

Son varias las ocasiones en que se ha intentado crear un lenguaje de programación que sirviera para todas las cosas y para todos los programa-

dores. Ninguno ha tenido éxito, pero el Pascal es sin duda el que más cerca ha estado de convertirse —al menos en el mundo del micro— en un auténtico lenguaje de programación universal. El Pascal fue diseñado por su creador, Nicholas Wirth, a raíz de su experiencia en la definición de otro lenguaje que también aspiraba a esa «universalidad», el Algol 68. Con el Algol 68, definido por un comité del que formaba parte Wirth, se pretendió crear el lenguaje polivalente, y el resultado fue algo muy extenso y complicado que no ha gozado de especial aceptación.

Tras esta experiencia, Wirth empezó a diseñar un lenguaje perfeccionado en el que se recogían muchas de las principales características del Algol 68. El resultado es un lenguaje de considerable potencia y fácil de «implementar» en ordenadores pequeños, incluidos los micros.

Con el Algol se pretendía que el programador pudiera diseñar un algoritmo y convertirlo fácilmente en un programa. El Pascal sigue el mismo principio, imponiendo al programa resultante una estructura muy rígida para impedir que el programador se aparte de esta norma. En consecuencia, el Pascal ha sido muy bien acogido en círculos académicos como herramienta pedagógica: los estudiantes aprenden buenos hábitos de programación sencillamente porque el Pascal les obliga a usar y desarrollar este particular principio. Además, Wirth trató de que su lenguaje fuera de lo más transportable, aunque posteriormente otros lo hayan ampliado y potenciando en perjuicio de su transportabilidad. Actualmente, la implementación en micro más difundida es la desarrollada por la Universidad de California en San Diego, llamada UCSD Pascal. Hay incluso un sistema operativo basado en esta implementación, el llamado UCSD p-System (véase Módulo 33).

Un programa en Pascal consta de un módulo principal y un conjunto de subrutinas o «procedimientos», a los que se llama desde el módulo principal o desde otro procedimiento. El Pascal, a di-

ferencia del C, impone estrictamente la regla de que no puede usarse ningún procedimiento que no se haya definido previamente. Esto viene a traducirse en que los programas siempre empiezan con el procedimiento de «nivel más bajo» y luego van sucediéndose los de niveles superiores hasta llegar al módulo principal, con el que concluye el listado. Este principio de ir «de abajo arriba» *(bottom-up)* es muy recomendable, porque obliga a organizar el programa de una forma clara y ordenada, aunque hay quien dice que cuando el módulo principal va al principio del listado es más fácil averiguar lo que hace el programa que cuando tenemos que empezar por pasar páginas y más páginas hasta dar con él.

Hay que definir todas las variables antes de empezar a usarlas, y declarar su tipo: REAL (números decimales) e INTEGER (números enteros). Esta declaración de variables se efectúa al comienzo del módulo:

```
VAR NUMBER, ERIC, TOTAL: REAL;
```

Con esta sentencia definimos como números reales las variables NUMBER, ERIC y TOTAL.

El Pascal cuenta con un buen conjunto de operadores. Además de los habituales, +, –, / (división) y * (multiplicación), encontramos el operador división entera, «div», que trunca siempre los decimales que puedan salir en una división, quedándose con el resultado entero (por ejemplo, 12/8=1,5 pero 12 div 8=1). El resto de una «div» se puede obtener con el operador módulo, «mod». Cuenta asimismo con las funciones trascendentes (seno, tangente y logaritmo, por ejemplo), y con otros operadores, como «odd» (que comprueba si un número es par o impar) y «round» (cuya función es redondear un número real al entero más próximo).

Hay en Pascal tres tipos de bucle, el primero:

```
FOR X:=1 TO 10 DO
BEGIN
  Y:= Y+6;
END
```

El bucle debe estar encerrado entre BEGIN y END para saber dónde empieza y acaba. TO se puede sustituir por DOWN TO, y se decrementa la variable de control en vez de incrementarla:

```
FOR X:= 10 DOWNTO 1 DO
```

En el bucle WHILE...DO se comprueba la condición antes de ejecutar el cuerpo del bucle:

```
WHILE X <> Y DO
BEGIN
  Y: = Y:+6;
END
```

mientras que en el bucle REPEAT...UNTIL la comprobación se efectúa al final:

```
REPEAT
  Y: = Y:+6;
UNTIL X = Y;
```

La entrada-salida es bastante flexible, siendo las principales instrucciones «writeln» (para escribir texto) y «readln» (para recoger datos del teclado).

El Pascal es apreciado, fuera del mundo académico, en las aplicaciones generales de gestión. Genera un código bueno y eficiente, y es fácil de aprender y de usar, pese a su quisquillosa insis-

tencia en las cuestiones sintácticas (constituye un buen ejemplo de lenguaje pensado para facilitarle la compilación al ordenador más que el uso al programador). También se presta a usos científicos de tipo general, pero no tanto para programación de sistemas, aunque en algunos casos se ha empleado.

Logo

Fue Seymour Papert (autor del vendidísimo *Mindstorms)* quien en la década de los sesenta sentó las bases del Logo, pensado fundamentalmente como un lenguaje de enseñanza para los niños. Firmemente enraizado en la psicología y la inteligencia artifical (véase Módulo 37), el Logo no es, sin embargo, tan sólo una herramienta para enseñar informática. Pretende introducir a los niños en el tema de la resolución de problemas con un ordenador, pero con el objetivo de enseñarles a abordar sistemáticamente la resolución de problemas en general.

En Logo es el niño el que «enseña» al ordenador, más que a la inversa, aprendiendo así a estructurar los procedimientos y, por extensión, a expresar la solución de un problema.

En el aula, el aprendizaje con Logo resulta aún más divertido gracias a la «tortuga». En un principio, esta tortuga es un robot consistente en una cúpula de plexiglás con ruedas. En el centro lleva una pluma retráctil con la que puede ir trazando una línea por la superficie sobre la que se mueve, normalmente una hoja grande de papel. La tortuga se conecta al ordenador por medio de un largo cable y sus movimientos se controlan tecleando distintas órdenes.

El Logo es un lenguaje interpretado, y las órdenes son obedecidas en cuanto el usuario las teclea. Sea, por ejemplo, la línea:

```
FORWARD 20
```

La pluma se desplaza hacia adelante 20 unidades de distancia (especificadas por el usuario).

```
RIGHT 90
```

Con esto torcemos 90 grados a la derecha. PENDOWN y PENUP sirven para bajar y subir la pluma respectivamente, para que la tortuga pueda trazar o no rayas al desplazarse. La secuencia:

```
PENDOWN
FORWARD 30
RIGHT 90
FORWARD 30
RIGHT 90
FORWARD 30
RIGHT 90
FORWARD 30
PENUP
```

hará que la tortuga dibuje un cuadrado.

La orden REPEAT permite al usuario repetir una acción el número de veces que se especifique. Pero, al igual que en Forth, en Logo se puede definir un conjunto de acciones, darle un nombre e incorporarlo a su vocabulario. Así, las líneas:

```
TO TRIANGLE SIDE
  REPEAT 3 [FORWARD : SIDE RIGHT 120]
END
```

definen un procedimiento llamado TRIANGLE, que permite dibujar un triángulo cuyos lados tienen la longitud especificada en SIDE. Si tecleamos TRIANGLE 50, dibujaremos un triángulo con lados de 50 unidades. El TO de la definición le dice al

Logo que la palabra siguiente es el nombre de un procedimiento, mientras que el SIDE denota que se va a usar una variable con ese nombre. La segunda línea contiene un bucle: se repite el número de veces que se especifique lo que va encerrado entre corchetes. El número de veces que se lleva a cabo la acción puede ser también una variable que se teclea junto al nombre del procedimiento,igual que se hace en éste con el 50. El :SIDE de dentro de los corchetes indica que hay que sustituir en ese punto SIDE por su valor, para que la tortuga sepa las unidades de distancia que debe desplazarse. RIGHT 120 hace que la tortuga gire 120 grados.

Esta posibilidad de extender las órdenes del Logo definiendo otras nuevas a partir de las ya existentes le dota de enorme flexibilidad. Algunos investigadores han llevado a cabo experimentos con sujetos de cuatro a dieciocho años de edad y se ha puesto en evidencia que todos, incluso los de menor edad, comprenden en seguida este mecanismo. Los de mayor edad eran capaces de definir procedimientos bastante complejos con mínima dificultad, consiguiendo crear dibujos notablemente intrincados con el Logo y la tortuga.

Naturalmente, todos los programas no funcionan a la primera, y esto le plantea al niño un nuevo reto. El primer intento de resolver el problema ha fracasado, ¿qué otras soluciones puede haber? ¿Cuál es el error? Estas preguntas incitan al niño a examinar las diversas alternativas en lugar de limitarse a buscar la solución «correcta». La idea es reemplazar los términos «éxito» y «fracaso» por el desafío de resolver el problema.

La tortuga-robot es un elemento importantísimo del aprendizaje del Logo, sobre todo en los niños pequeños. La experiencia demuestra que la aceptan fácilmente, tomándola en ocasiones por una mascota. Sin embargo, es un dispositivo lento y que exige mucho espacio (aparte de gastar cantidades ingentes de papel). Además suele ser caro, lo que constituye su principal inconvenien-

te. Por esta razón se usan mucho las «tortugas de pantalla». Especialmente populares en los ordenadores domésticos, revisten la forma de un triangulito que aparece en la pantalla y que el usuario puede desplazar casi como si se tratara de una tortuga «real». Esta alternativa no sólo es más rápida, barata y económica (sobre todo en papel), sino que permite borrar cualquier zona que se quiera, abriendo la posibilidad de dotar de cierto grado de animación al dibujo de la pantalla.

El Logo, en cualquier caso, tiene muchas otras posibilidades aparte de la de trazar rayas. Su relación con la inteligencia artificial se pone de relieve en sus cualidades para el manejo de listas, y en algunas implementaciones se dispone de ciertos recursos gráficos que permiten al usuario definir *sprites* (caracteres gráficos especiales) para hacer dibujos en la pantalla.

SOFTWARE DE SISTEMAS

Para que un ordenador haga algo es preciso proporcionarle una lista de instrucciones llamada programa. Cambiando el programa podemos conseguir que el ordenador lleve a cabo una tarea nueva y del todo diferente.

Los sistemas operativos

Casi todos los programas contienen una serie de operaciones comunes. Es muy corriente, por ejemplo, que en determinado momento sea preciso introducirles alguna información desde el teclado, y esa información habrá que mostrarla en la pantalla para que el usuario pueda verificar que la respuesta tecleada es correcta. Hay que programar el ordenador, por tanto, para recoger caracteres del teclado y escribirlos en pantalla.

Sería una pérdida de tiempo tener que repetir este tipo de instrucciones cada vez que se escribe un programa. Aunque la gestión del teclado y la pantalla sea relativamente sencilla en muchos ordenadores, las operaciones de manejo de discos son ya más complejas. Además, el modo de hacerlo varía de un ordenador a otro, y sería preciso reescribir grandes porciones de los programas cada vez que se quisiera ejecutarlos en un tipo de máquina diferente.

Para resolver este problema de coherencia se ha ideado un tipo especial de *software* llamado «*software* de sistemas» o «sistemas operativos».

Se trata de una colección de subrutinas que se encarga de todas las funciones básicas de «buen gobierno», tales como la gestión de la pantalla y el teclado, o la de los discos si procede. Cuando una firma quiere fabricar una máquina que utilice un sistema operativo de los conocidos, basta modificar las partes del programa que sean dependientes de la máquina. Según estén diseñados el ordenador y el sistema operativo, esto puede reducirse incluso a un mero cambio en las direcciones de los puertos de entrada-salida. Lo único que tiene que hacer un programador que quiera sacar datos de un fichero o escribir un carácter en la pantalla, por ejemplo, es activar la subrutina apropiada del sistema operativo. Una vez que ésta ha cumplido su misión, el ordenador sigue con el programa.

Aparte del ahorro de tiempo y esfuerzo que supone el uso de un sistema operativo de los conocidos, queda garantizado que cualquier programa escrito para tal sistema funcionará correctamente en todos los ordenadores equipados con él.

El CP/M

El sistema operativo más usado en microordenadores es el CP/M —*Control Program/Microcomputer*—, creado por Digital Research en los Estados Unidos. En un principio, el CP/M fue creado para los ordenadores basados en la CPU 8080 de Intel (véase Módulo 17). Luego, el 8080 fue reemplazado por el *chip* Z80 de Zilog, que es mucho más potente pero incluye también el juego de instrucciones binarias del 8080 como parte del suyo propio. Por ello, el CP/M, que al comienzo del auge de los micros era el único sistema operativo normalizado, también se pudo usar en los sistemas basados en el Z80, con lo que se popularizó mucho. Aunque popular, el CP/M no resulta especialmente «amistoso» para el usuario. Es, no obstante, un sistema estándar, y hay cantidades ingentes de *software* escrito para él.

Los sistemas operativos desempeñan un impor-

tante papel en la microinformática, donde el usuario tiene un control absoluto y directo sobre la máquina y suele tener pocos o nulos conocimientos de informática. El sistema operativo suele ser lo primero con que se encuentra el usuario al encender el ordenador. Actualmente, los sistemas operativos de los microordenadores están aún lejos de proporcionar al usuario una conexión clara, útil y autoinformativa con el *software* de aplicaciones (véase Módulo 35) que esté utilizando.

Por ejemplo, cuando se enciende un sistema CP/M y se carga el sistema operativo, lo normal es que aparezca en pantalla tan sólo un breve mensaje de *copyright* y, en la línea siguiente, "A>". Esto de por sí dice muy poco si no se le explica que la "A" significa que el ordenador está utilizando en ese momento la unidad de disco A y que con el ">" el CP/M le está diciendo que puede teclear una orden. Este mensaje "A>" recibe el nombre técnico de "aviso del sistema» *(system prompt).*

El CP/M tiene dos tipos de órdenes. Unas están incorporadas al sistema operativo mismo —forman parte del programa CP/M en RAM— y permiten realizar varias tareas sencillas. Por ejemplo, se puede pasar a otra unidad de disco tecleando "B:" y pulsando a continuación RETURN. Lo normal es escuchar después en la unidad B un «clic» y un zumbido y recibir el «aviso» que indica que nos encontramos en el disco B: "B>". Tecleando "DIR" y luego RETURN veremos en pantalla el listado del directorio del disco que estemos usando. El CP/M no indica el tamaño de los ficheros ni el espacio que queda libre en el disco. Si estamos en el disco A y queremos ver lo que hay en el disco B sin cambiar de unidad, basta teclear "DIR B:"

Hay otras órdenes de CP/M que se presentan en forma de programas independientes que residen en el disco y se ejecutan cuando así se solicita. Entre ellos figuran las utilidades que efectúan operaciones como «formatear» un disco nuevo (véase Módulo 22), copiarle el CP/M, y copiar fi-

¿COMO FUNCIONA EL CP/M?

El CP/M es un programa que reside en una zona reservada del disco. Una ROM procede, al encender el ordenador, a cargar automáticamente en la RAM el sistema operativo y luego le cede control, tras lo que el CP/M escribe en la pantalla su famoso aviso "A>" y espera a que el usuario teclee una orden.

El diagrama muestra la organización de la memoria de un sistema típico basado en el CP/M de 8 bits, con 64 koctetos. Se le llama «mapa de memoria», y en él se aprecia la compartimentación de la memoria y los diversos usos a que se destina cada parte. Los números de la derecha son las direcciones hexadecimales de comienzo de las secciones. La ROM reside fuera de la memoria principal del sistema. Al encender la máquina se desconecta una parte de la RAM y se conecta la ROM y una vez cargado el CP/M, se desconecta la ROM, que ya no se va a usar.

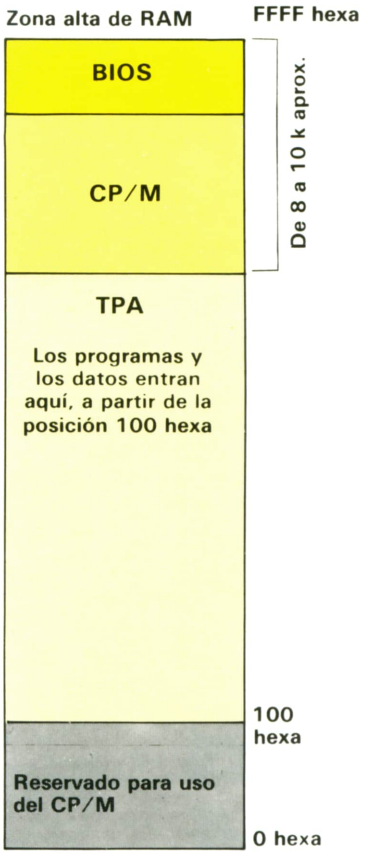
En la parte parte baja del mapa hay una pequeña zona, que se extiende de la posición 0 hexa a la FF hexa, reservada para las necesidades del CP/M. Este reside en la zona más alta de memoria. Cuando se instala el CP/M en un sistema nuevo, ciertas partes hay que «particularizarlas» para el sistema. El grado de particularización depende de ciertos detalles del *hardware*: puede tratarse sólo de cambiar la dirección de un puerto de E/S o exigir otras adaptaciones. En cualquier caso, hay un conjunto de funciones normalizadas que es preciso que existan. Estas funciones facilitan todas las operaciones básicas de E/S del sistema, tales como recoger caracteres tecleados, escribir caracteres en pantalla, etc. Esta sección del CP/M se llama BIOS *(Basic I/O System)*.

El resto del CP/M se puede usar tal cual, y reside en la memoria que queda debajo del BIOS. En esta zona se encuentran el Procesador de las Ordenes de Consola o CCP *(Console Command Processor)* y el Sistema Operativo Básico de Disco o BDOS *(Basic Disk Operating System)*, que gobierna el almacenamiento en disco de la información. La cantidad de memoria que ocupa todo el CP/M depende del tamaño del BIOS, y suele oscilar entre 8 y 10 koctetos.

A la memoria comprendida entre la posición 100 hexa y el comienzo del CP/M se le llama Zona Temporal de Programas o TPA *(Transient Program Area)*. En esta zona es donde el CP/M

cheros de un disco a otro. Un programa que resida en el disco que estamos usando en un momento dado se ejecuta sin más que teclear su nombre: el CP/M lo carga en memoria y lo arranca.

En comparación con otros sistemas operativos, el CP/M es fácil de usar. Aunque tiene sus rare-



carga los programas, y en lo que queda libre es donde el programa almacena sus datos. Una vez arrancado el CP/M todo lo que se le teclee lo considera una orden. El CCP va recogiendo los caracteres tecleados y colocándolos en una zona reservada de memoria hasta que recibe un RETURN, señal de que la orden ha concluido. Entonces, el CCP examina los caracteres tecleados para ver si se trata de una orden «intrínseca» (que forma parte del CP/M), tal como DIR. De no serlo, el CCP supone que se trata de un programa y el BDOS trata de localizarlo, cargarlo y ejecutarlo. Si existe el programa, se carga en RAM a partir de la posición 100 hexa, y el CP/M ordena al procesador ir a esa posición y ejecutarlo. Si el programa no está en el disco que se está usando, el CP/M vuelve al escribir en pantalla la orden seguida de un «?» y después, de nuevo, el «aviso».

Para que el CP/M pueda distinguir un programa de, pongamos, un fichero de texto, todo lo que haya en el disco tiene que tener un nombre de hasta 8 caracteres más una extensión de hasta 3 caracteres separada por un punto del nombre principal. Podemos, por ejemplo, darles a todos los ficheros de texto la extensión «LET» y a los programas la «COM». Para activar un programa hay que teclear el nombre (pero no la extensión) y pulsar RETURN.

zas —por ejemplo, la utilidad que copia ficheros de un disco a otro se llama "PIP" en vez de "COPY"—, casi todos los usuarios llegan a dominarlo en poco tiempo. Su principal inconveniente es que cuando algo va mal, apenas da pistas ni explicaciones. Supongamos que queremos copiar un

fichero de un disco a otro sin meter primero físicamente un disco en la segunda unidad. El CP/M escribirá en la pantalla el mensaje «BDOS ERR: BAD SECTOR», que por sí sólo dice bien poco y aparece en cuanto hay algún problema.

Variantes de este sistema operativo

Se ha intentado en varias oportunidades mejorar el CP/M con desigual éxito. Una compañía llamada Cromemco ha creado para sus máquinas un sistema operativo inspirado en CP/M llamado CDOS *(Cromemco Disk Operating System).* Este sistema es mucho más «amistoso» que el CP/M. Es posible, por ejemplo, «proteger contra escritura» un disco flexible tapando con una lengueta una muesca que llevan en la funda protectora. De este modo se pueden leer los ficheros de un disco y ejecutar los programas que contenga, pero no escribir nada en él, lo que elimina el peligro de alterar o destruir accidentalmente la información que contenga. En CP/M tratar de copiar un fichero a un *floppy* protegido contra escritura ocasionaría la aparición del famoso mensaje "BDOS ERR ...". CDOS, sin embargo, nos brinda uno mucho más informativo: *Diskette in drive B is write-protected* («El disco flexible de la unidad B está protegido contra escritura»).

También se han creado varios programas «de menú» para facilitar el manejo del CP/M. La idea es que al encender el sistema, en lugar del escueto «aviso» aparece en pantalla una lista de todas las órdenes y programas que contiene el disco en uso (el «menú»), lista de la cual podremos seleccionar una opción tecleando su número o llevando el cursor hasta ella y pulsando RETURN.

El CP/M es el sistema operativo más usado, pero no el único. Sólo funciona en máquinas que lleven determinada CPU (véase Módulo 18). Sin embargo, algunos microordenadores pueden convertirse al CP/M añadiendo una tarjeta suplementaria con un procesador Z80 y, normalmente, algo más de RAM. El Apple II, por ejemplo, está basa-

do en el procesador 6502, cuyo juego de instrucciones es diferente al del 8080/Z80, y es incapaz por tanto de ejecutar el CP/M.

Compatibilidad de sistemas

Cuando no había más microprocesadores que los de 8 bits, la cuestión de los sistemas operativos era sencilla, pero desde que aparecieron los de 16 las cosas se han complicado notablemente. El primer problema consiste en que los fabricantes de *chips* de 16 bits no se toman la molestia de garantizar que éstos funcionen con los códigos de instrucciones binarios de 8 bits. Zilog, por ejemplo, que dominaba el mercado de los 8 bits con su Z80, ha creado un *chip* de 16 bits muy potente llamado Z8000, pero ni siquiera ha intentado incluir el juego de instrucciones del Z80 en el Z8000, perdiendo como consecuencia un sector del mercado potencialmente valioso. De igual manera el Intel 8088, actualmente el más popular de los *chips* de 16 bits, es incapaz de ejecutar el *software* escrito para el antiguo procesador de 8 bits, el 8080.

Sistemas operativos de 16 bits

Esta falta de compatibilidad ha tenido, sin embargo, un resultado positivo. Como buena parte del *software* escrito para las máquinas de 8 bits no vale para las de 16, las compañías de *software* de sistemas han tenido que crear productos nuevos, y existen ya varios para los micros de 16 bits. Por desgracia, todavía no ha aparecido ningún sistema operativo verdaderamente adecuado para el creciente ejército de usuarios no especialistas en informática. Tres sistemas operativos compiten actualmente en el mercado del micro de 16 bits: el CP/M-86 (la versión para 16 bits del CP/M), el MS-DOS y el Unix. Hay un cuarto competidor, el UCSD p-System, algo más rezagado.

El CP/M-86 es, para el usuario, prácticamente idéntico al sistema operativo de 8 bits CP/M. Pero no es posible usar *software* escrito para CP/M

LA FAMILIA CP/M

Los ordenadores basados en el CP/M son los dueños del mundo de la microinformática comercial. Existe toda una familia de sistemas operativos CP/M derivados del original para 8 bits nacido a finales de la pasada década. La existencia de un sistema operativo estándar contribuyó a potenciar el crecimiento de la industria del micro.

CP/M-80. Nombre actual de la versión original del CP/M para 8 bits. El sistema operativo ha experimentado varias revisiones desde su introducción, y la versión más corriente ahora es la 2.2. Es el de 8 bits más usado en todo el mundo.

CP/M Plus. Llamado en un principio CP/M 3, es la última versión del CP/M de 8 bits, pero está aún poco difundida. Incorpora mejoras considerables respecto al CP/M-80 y es mucho más «amistoso». Al popularizarse los micros de 16 bits cabe esperar que los de 8 se abaraten, y el CP/M Plus tendrá el mismo éxito que sus predecesores.

CP/M-86. Es la versión para 16 bits del CPM-80, muy popular en la nueva generación de micros de 16 bits. Desde el punto de vista del usuario, el CP/M-86 es prácticamente idéntico al CP/M-80.

de 8 bits con el CP/M-86, por lo que no se ve claro el sentido que tiene haber conservado una fachada tan poco «amistosa» para el usuario como la del CP/M. El CP/M-86 funciona con los micros basados en el 8088 y, por tanto, también con los que usen el *chip* 8086, compatible con el anterior pero más potente.

El MS-DOS es creación de la compañía Microsoft. Llamado inicialmente PC-DOS, sistema operativo para el ordenador personal de IBM, hoy disponen de él casi todos los ordenadores basados en el 8088 o el 8086. A primera vista, el MS-DOS se parece al CP/M-86, pero en realidad son muy distintos en funcionamiento y en uso. Es más «amistoso» que el CP/M-86, siendo sus órdenes más fá-

MP/M-80 y **MP/M-86.** Son versiones «multiproceso» del CP/M-80. El multiprocesamiento es un truco de *software* para repartir la atención de la CPU entre tareas diferentes y llevar a cabo varios trabajos a un tiempo. Esta técnica da buenos resultados en ordenadores grandes y potentes, pero no tanto en los micros, sobre todo en las máquinas de 8 bits, en las que el procesador no tiene suficiente potencia para sacarla adelante.

CP/M-86 concurrente. Los procesadores de 8 bits no pueden con el multiprocesamiento, pero con los de 16 hay más esperanza. A pesar de que es más sensato técnica y económicamente darle a cada usuario su CPU, hay veces en que conviene realizar más de una tarea simultáneamente (imprimir un fichero de texto mientras se edita otro, por ejemplo), y a tal objeto se creó el CP/M-86 concurrente. Es pronto para decir si tendrá o no éxito, pero es una idea más sensata que los sistemas de micros multiusuario.

CP/NET. Este sistema sirve para enlazar un ordenador a una red. Cada usuario tiene su pantalla, su teclado, su memoria y su CPU, pero todas las máquinas están enlazadas para compartir los recursos caros, por ejemplo, discos duros o impresoras. Este enlace exige una *interface* para gestionar las comunicaciones por toda la red, y éste es el papel de CP/NET.

ciles de usar y sus mensajes de error más informativos. Una de las desventajas del MS-DOS es que, salvo que tengamos un PC de IBM, la gama de *software* disponible es reducida en comparación con el CP/M-86. Hasta ahora, el PC de IBM se ha vendido bien en Estados Unidos, pero no en Europa. Debido a ello, se ha llegado a una situación en la que los usuarios norteamericanos de MS-DOS cuentan con gran cantidad de *software* para sus IBM, mientras que los europeos, que usan otras máquinas de 16 bits, se encuentran prácticamente confinados a los paquetes de CP/M-86. De todos modos, las cosas evolucionan tan deprisa que esta situación podría cambiar pronto.

Motorola produce un microprocesador de 16

bits considerado el más potente del mercado, el 68000. El juego de instrucciones del 68000 es, como era de temer, totalmente distinto del de los 8088 y 8086, y requiere por tanto un sistema operativo y un *software* diferentes. La mayoría de los sistemas basados en el 68000 utilizan un sistema operativo llamado Unix o variantes del mismo. Con el Unix se pretendió dotar a los programadores de un «ambiente» potente y de fácil manejo para escribir *software*. Es un sistema operativo muy versátil para el profesional de la informática pero, por desgracia, poco adecuado para el aficionado o el usuario de gestión, que son los que más usan los microordenadores. Es posible dotar al Unix de una fachada más «amistosa», y algunos fabricantes así lo han hecho, pero su verdadero problema es que apenas cuenta con *software* de aplicaciones. Diseñado para que personas que trabajan en un entorno fundamentalmente técnico puedan desarrollar su *software*, no se le ha dedicado a las aplicaciones comerciales para las que fueron específicamente creados otros sistemas operativos. El Unix se utiliza sobre todo en universidades, escuelas técnicas y compañías de *software;* por ello no parece probable que llegue a popularizarse entre los no expertos.

El UCSD p-System se distingue de los anteriores en que vale para cualquier ordenador, pudiendo ejecutarse sin ningún cambio en una máquina los programas escritos con él para otra. Esto se consigue mediante el uso de un «seudo-código». A diferencia de los compiladores usados en otros sistemas operativos, los compiladores del p-System no generan código de máquina para un procesador concreto, sino un código intermedio muy eficiente que luego, al ejecutarse el programa, traduce el núcleo del sistema —el traductor de código p—. Para instalar un p-System en un procesador nuevo lo único que hay que hacer es reescribir este núcleo traductor en el código de máquina de la nueva CPU.

Son muchas ya las máquinas para las que hay un p-System, que tiene además la ventaja de garantizar totalmente la «transportabilidad» del *software,* pero pese a ello todavía no está plenamente aceptado por la mayoría de las compañías de *software.* Quizá se deba a que al usuario le parece un sistema pensado más para el programador que para el no experto. Sus partidarios más entusiastas se cuentan sin duda entre los programadores de sistemas, lo que para el mercado de los aficionados no es un buen síntoma.

BASES DE DATOS

Una de las aplicaciones más importantes y de expansión más acelerada en la informática de nuestros días es el almacenamiento de información. En los ordenadores se guarda información de todo tipo: expedientes fiscales, información sobre salubridad y empleo, datos financieros etc.

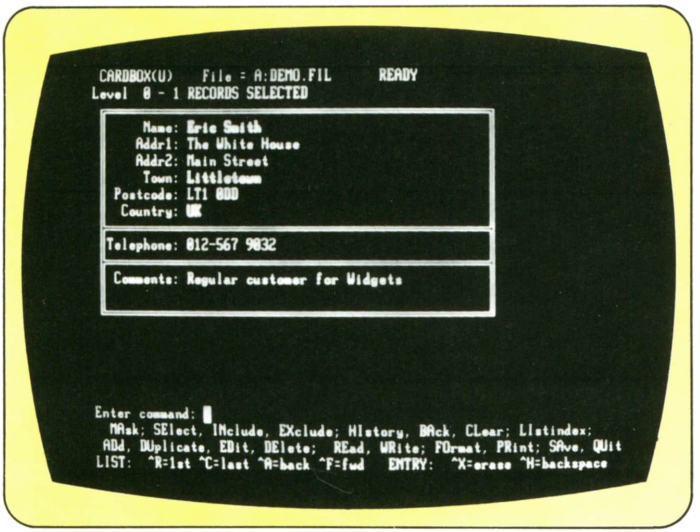
Sistemas gestores de bases de datos

No tendría ningún sentido meter toda esta información en un ordenador si no se pudiera recuperar luego fácilmente. Y, además, ya puestos a usar un ordenador, lo lógico es aprovechar todas sus posibilidades para clasificar datos, comparar cosas y tomar decisiones. Existen ya muchos programas capaces de realizar este «archivo inteligente» de la información, los llamados «sistemas gestores de bases de datos».

Estos sistemas nos permiten introducir estructuradamente información. Almacenarla en el ordenador, y luego recuperarla y examinarla. Para entender el funcionamiento de uno de estos sistemas conviene repasar algunos ejemplos de aplicaciones corrientes en el mundo del micro.

Supongamos que somos dueños de una pequeña empresa con mil clientes extendidos por una amplia zona geográfica. Supongamos que hemos decidido ampliar nuestra línea con un nuevo producto (el Modelo IV), y queremos comunicárselo a nuestros clientes con la esperanza de que alguno de ellos nos lo compre. En esta situación, po-

CARDBOX



El Cardbox es un sistema de semibase de datos de bajo coste. Utilizando unas órdenes sencillas, el usuario puede «dibujar» el equivalente electrónico de una ficha en la pantalla, subdividirla y poner un nombre a cada subdivisión. Cardbox permite buscar en las fichas a gran velocidad, hasta dar con la que contenga una palabra especificada por el usuario. Sin embargo, no tiene la flexibilidad de permitirnos buscar por varias «claves» simultáneamente. Esta desventaja queda compensada por lo sencillo de usar que resulta. Es un sistema ideal para el no experto, porque no se necesita aprender un gran número de conceptos informáticos para sacarle pleno rendimiento. Otra ventaja es que no hace falta estar familiarizado con el lenguaje informático.

---

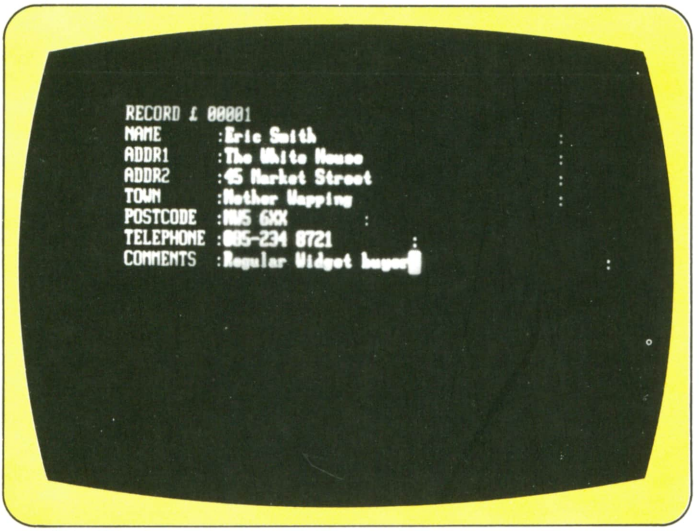
dríamos preparar una circular, imprimir mil ejemplares y enviar uno a cada cliente. O también, utilizando un ordenador y un sistema de bases de datos, incluir más información y crear una carta más perfeccionada y útil:

Nombre: Juan López.
Dirección: Calle Ancha.
Ciudad: Villanueva.
Profesión: Contable.
Apelativo en el encabezado de las cartas: Juan.
Primera compra: 1976. Artículo: Modelo II.
Ultima compra: 1983. Artículo: Modelo III.

Habrá bastantes personas que, como Juan López, hayan comprado recientemente otro modelo de este producto y, por tanto, no tendrán excesivo interés en comprar uno nuevo ahora. Además no para todos los clientes tendrá interés o aplicación un producto de este tipo: sabemos, por ejemplo, que en la base de datos hay unos cuantos fontaneros, y los fontaneros nunca lo compran, por lo que carece de objeto mandarles una carta.

Así que, encendido ya el ordenador, ponemos el disco que contiene el programa de base de datos en una de las unidades y el disco con la base de datos propiamente dicha en la otra y, mediante la utilidad de búsqueda *(search)* que tendremos, preguntamos por las personas que no han comprado este producto durante los últimos seis meses y que no son fontaneros. El ordenador consultará todos los registros de la base de datos, examinando la profesión de cada persona y excluyendo a los fontaneros y a los compradores de los últimos seis meses. Los que queden una vez terminado este proceso de selección son los posibles interesados, así que le pedimos al ordenador que imprima sus nombres y direcciones. Se puede usar luego otro programa llamado «procesador de textos» (véase Módulo 36) para escribir cartas aparentemente personalizadas para cada cliente.

Ahora imaginemos que somos unos fotógrafos entusiastas, dueños de una colección de miles de diapositivas acumuladas a lo largo de los años. Tratar de encontrar una imagen determinada nos supondría horas de valioso tiempo y mucho trabajo. Ahora bien, si todas tienen su número y un registro en la base de datos, si queremos una foto de, pongamos, la tía María, basta solicitar de la base de datos la lista de los números de todas las diapositivas en las que sale dicha persona. Se puede sin embargo concretar más y pedir los números de referencia de las realizadas entre ciertas fechas o en ciertas situaciones. El único factor limitativo es aquí el volumen de la descripción que asignamos a cada registro de la base de datos.



dBASE II quizá sea el sistema de base de datos para micros más vendido. Permite manipular bases de datos de tamaño pequeño y medio, mediante órdenes inspiradas en el inglés. Es extremadamente potente, pero al precio de ser menos «amistoso» y más difícil de usar. A diferencia del Cardbox, exige del usuario el conocimiento de varios términos y técnicas utilizadas en informática, y es más difícil modificar el formato de los registros en la base de datos una vez establecido y en marcha. La potencia y la flexibilidad del dBASE II se acrecientan gracias a su lenguaje de programación incorporado. Esto permite construir aplicaciones complicadas y el resultado es un sistema fácil de usar e imposible de estropear, apto para los usuarios más novatos.

Distintos tipos de base de datos

Teniendo en cuenta el nivel actual de las bases de datos, los ejemplos anteriores son aplicaciones sumamente simples que casi cualquier sistema podría gestionar sin dificultad. Hay sistemas para micros básicos y otros más rebuscados. Hay sistemas con órdenes tan complejas que para aprovechar plenamente su potencia hay que aprender casi un nuevo lenguaje de programación.

Como ejemplo de un sistema básico citaremos el «Cardbox». Permite «dibujar» en la pantalla una «ficha», dividirla en varias zonas, poner nombre a

cada zona, y luego introducir información que va a parar a un fichero en disco. Sus principales virtudes son la facilidad de uso y la flexibilidad. Si, por ejemplo, una vez establecida la base de datos nos damos cuenta de que no nos gusta como salen las cosas por pantalla, se puede cambiar fácilmente el formato o crear uno nuevo en el que aparezca sólo parte de cada registro. «Cardbox», sin embargo, no es una verdadera base de datos, pues no se puede clasificar su contenido por orden alfabético, por ejemplo, ni hacer operaciones matemáticas con los datos.

Pero hay productos, como dBase II, que permiten hacer casi cualquier cosa con la información. dBase II cuenta con su propio lenguaje de programación, mediante el cual se pueden poner a punto sistemas muy complejos y enteramente automáticos que puede usar cualquier persona. Es posible tener todos nuestros archivos comerciales en una base de datos dBase II y acceder a la información para diferentes necesidades: facturación, previsiones de liquidez, contabilidad general y direcciones de clientes, por ejemplo.

HOJAS DE CALCULO

Las hojas de cálculo se están convirtiendo en un componente indispensable de la microinformática de gestión. La hoja de cálculo es un «paquete» de *software* que facilita la ejecución de los cálculos comerciales que normalmente se efectuarían con papel, lápiz y una calculadora. Imaginemos que la pantalla de nuestro monitor es una especie de ventana y que a través de ella contemplamos una hoja grande de papel. Esta hoja está subdividida en una serie de rectángulos que la cuadriculan. Los rectángulos reciben el nombre de «celdas» y cada celda puede contener datos y cifras relativas a nuestra empresa.

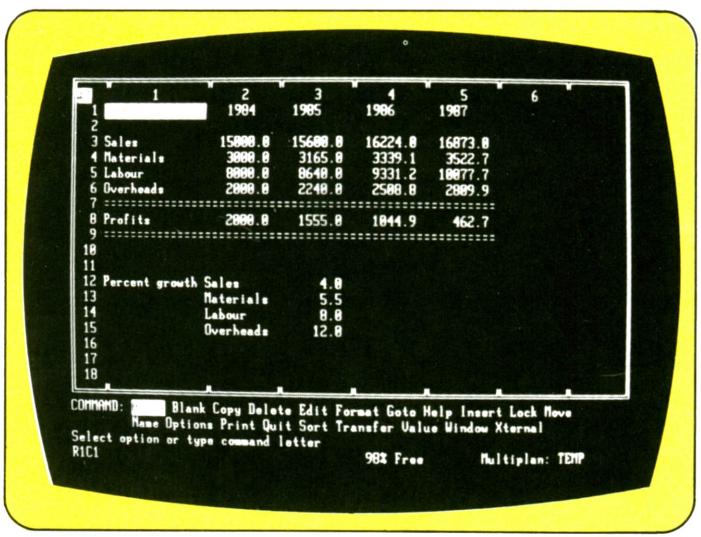
Las hojas de cálculo en la práctica

Para ver qué aplicación tienen las hojas de cálculo, imaginemos que somos dueños de una fábrica. El cálculo de los objetivos de ventas forma parte integrante de éste y de muchos otros tipos de empresa. También puede interesarnos saber lo que deben aumentar anualmente los ingresos procedentes de las ventas para que no disminuyan los beneficios pese al incremento del coste de las materias primas, los salarios y demás gastos, tales como la calefacción y la energía eléctrica.

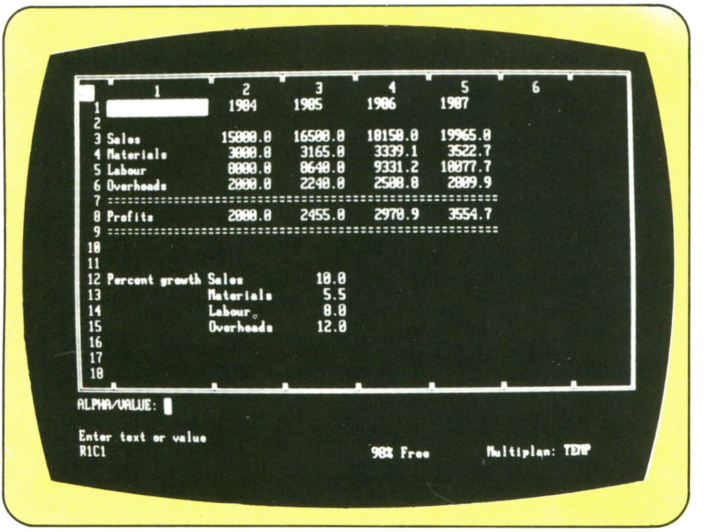
Nuestro experto financiero prevé una subida del 5% anual en el coste de los materiales, un 8 en salarios y un 12 en gastos generales. Con estas ci-

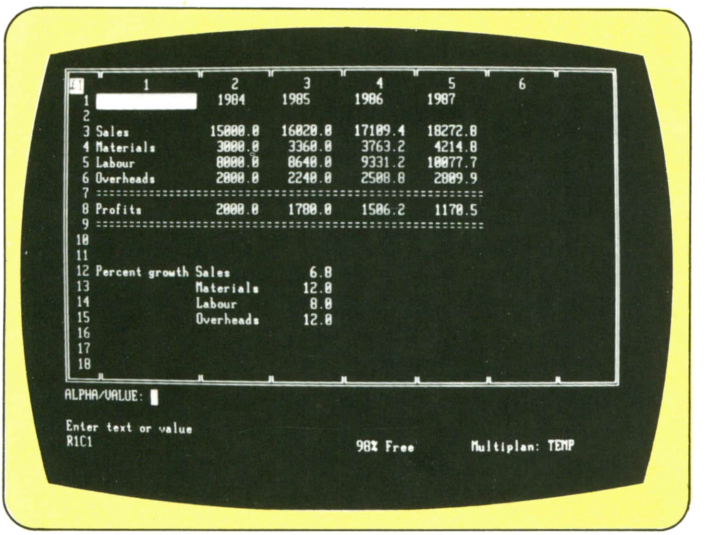
EL USO DE LAS HOJAS DE CALCULO

Permiten experimentar con grandes cantidades de datos interrelacionados. En este ejemplo se calculan los beneficios esperados, restando los costes de las materias primas, los salarios y los gastos generales de los ingresos por ventas. Se fijan ciertos aumentos en los costes y salen en pantalla sus efectos sobre el beneficio. Para contrarrestar las pérdidas resultantes, las ventas habrán de tener un crecimiento anual del 10 por ciento (derecha, arriba). Esto tendrá un efecto magnífico sobre el beneficio, que será el doble hacia 1988. ¿Qué sucedería, sin embargo, si las ventas sólo crecieran un 6,8 por ciento, y el coste de los materiales un 12 por ciento en lugar del 5,5 por ciento? No hay más que modificar estas cifras (derecha, abajo) para observar los efectos sobre el beneficio. Ajustando las cifras hasta obtener el beneficio deseado, se puede llegar a la cifra objetivo de crecimiento de las ventas.



fras y una calculadora seguro que en pocas horas llegaríamos a la conclusión de que es preciso un crecimiento de las ventas del orden del 4% para que el margen de beneficio siga siendo acepta-





ble. Pero con este sistema de cálculo manual, cualquier modificación que se quiera introducir en alguno de los supuestos implica volver a calcularlo todo.

Un «paquete» informático de hoja de cálculo, sin embargo, es mucho más flexible. El aspecto externo de la hoja de cálculo puede ser análogo al de la hoja de papel utilizada en el cálculo manual, pero ahí termina el parecido. Cada vez que cambia alguno de los factores que afectan al rendimiento de nuestro negocio, lo único que hay que hacer es llevar el cursor a la celda llamada, por ejemplo, «crecimiento de salarios», y teclear el nuevo porcentaje. El ordenador recalculará de inmediato todos los valores de todas las celdas que se vean afectadas por el cambio. De esta manera podemos conocer al instante y con toda exactitud nuestra probable situación financiera en algún momento del futuro, utilizando la información más reciente de que se disponga. La hoja de cálculo permite también ensayar las cifras más altas y más bajas previsibles para las distintas variables, ver cómo afectan a los beneficios y observar el efecto de aumentar en varios puntos el crecimiento de las ventas.

El funcionamiento de la hoja de cálculo

Las hojas de cálculo nos permiten permiten establecer una matriz de celdas y definir las relaciones que ligan los distintos elementos. Luego, al modificar el valor de una celda se modificarán de modo automático todas las demás celdas, ligadas a ella por una fórmula matemática. Esta flexibilidad nos permite hacer especulaciones del tipo «¿y qué ocurriría si...?», visualizando de inmediato en pantalla los resultados del experimento.

La mayor parte de las hojas de cálculo existentes permiten guardar en disco los datos, imprimirlos, tratarlos con un procesador de textos e incluso convertir la información numérica en gráficos o histogramas en los que normalmente resulta más intuitivo identificar las diferencias que cuando se presentan, sumergidas en un mar de cifras.

Probablemente, el programa de hoja de cálculo más conocido y de más éxito sea el VisiCalc, creado para el Apple II. Este *software* propició un con-

siderable número de ventas del ordenador Apple II, pues había quien estaba dispuesto a comprar el *hardware* exclusivamente para poder trabajar con él. A la vista del éxito del VisiCalc, el resto de los encartados en el negocio del *software* se dieron cuenta de que esto daba dinero y aparecieron al poco tiempo docenas de programas de hoja de cálculo, la mayor parte de los cuales decían ser superiores al VisiCalc, fuera en facilidad de uso, posibilidades o potencial matemático. En la industria informática muy pronto se empezó a llamar a estos paquetes «Visi Clones».

El VisiCalc fue responsable de la venta de gran cantidad de *hardware,* pero también, al menos parcialmente, de la infiltración de los microordenadores en muchas grandes empresas. Los ejecutivos, frustrados por la inflexibilidad de sus departamentos de informática, cayeron en la cuenta de que podían obtener lo que necesitaban más deprisa y con menos coste en su propia mesa de despacho con un microordenador que recorriendo los cauces oficiales.

Aplicaciones de las hojas de cálculo

Hoy día, casi todos los ordenadores de gestión existentes en el mercado cuentan con su hoja de cálculo, y las hay incluso, en versiones simplificadas, para algunos micros domésticos. El uso del color, cada vez más generalizado, dota de una nueva dimensión a las hojas de cálculo (y a otros paquetes de *software),* permitiendo, por ejemplo, que las cantidades negativas de dinero aparezcan destacadas en rojo.

Sería erróneo, sin embargo, suponer que las hojas de cálculo sólo sirven para trabajos financieros. Asombra la cantidad de aplicaciones que tienen: casi cualquier cosa que exija la manipulación de cifras ligadas por relaciones complejas.

PROCESAMIENTO DE TEXTOS

Los ordenadores se han venido aplicando al almacenamiento y la manipulación de textos desde hace mucho tiempo, pero, hasta el auge de los micros, en forma muy limitada, porque emplear un ordenador grande y costoso para escribir cartas de promoción comercial, artículos de revista o libros no parecía económicamente sensato.

El impacto de los micros

Ahora, con los micros resulta ya perfectamente razonable comprar un ordenador destinado exclusivamente a la manipulación de textos, y muchas firmas venden máquinas «dedicadas» a esta tarea e inservibles para cualquier otro tipo de aplicación. El procesamiento de textos, posible hoy con cualquier micro de gestión, y con muchos domésticos también, constituye ya una de las aplicaciones principales de los ordenadores pequeños. ¿Qué significa pues procesamiento de textos, y cuáles son sus ventajas e inconvenientes?

El procesamiento de textos en la práctica

El procesamiento de textos significa, en el nivel más rudimentario, teclear un texto en el teclado de un ordenador (véase Módulo 20), poder almacenarlo en disco (véase Módulo 22) o cinta (véa-

se Módulo 21), poder recuperarlo en una fecha posterior y escribirlo en un papel mediante una impresora (véase Módulo 23).

Pero, en realidad, son mucho más versátiles. Para empezar, una vez tecleado el texto podemos volver a verlo en la pantalla y corregir posibles errores mecanográficos. Como el texto está en la memoria del ordenador, esas modificaciones no suponen más que cambiar el contenido de las posiciones de memoria adecuadas. Análogamente, si no nos gusta la redacción de una frase, la podemos cambiar, pulir, y trabajarla cuanto nos plazca hasta quedar satisfechos.

Más aún, puede que el orden en que hemos escrito el texto no sea el adecuado. No hay que preocuparse. Basta colocar el cursor al principio del párrafo afectado, pulsar una tecla o código de control, llevar el cursor al final del párrafo, y pulsar otra tecla que indique al ordenador que queremos hacer algo con todo ese bloque de texto. Si hay que borrarlo, el ordenador lo suprimirá. Si está donde no debe, llevamos el cursor al sitio donde queramos ponerlo, pulsamos otro código, y todo el bloque se trasladará a su nueva posición (reordenándose convenientemente el resto del texto).

Supongamos, por ejemplo, que estamos escribiendo un largo artículo en el que aparece con frecuencia la palabra «Los Angeles». Es una pérdida de tiempo tener que teclearlo una y otra vez, pero no queremos utilizar abreviaturas. La solución es escribir «LA» mientras tecleamos el artículo y luego, volver al principio del texto, y pulsar el código o tecla de la orden de «búsqueda y sustitución», con lo que se activa uno de los recursos más cómodos del procesamiento de textos. El ordenador preguntará «¿Sustituir qué?», y tecleamos «LA»; «¿Por qué?» preguntará luego, y contestaremos «Los Angeles». Pulsando la tecla RETURN veremos de inmediato que todos los «LA» que aparecían en el texto van siendo sustituidos por «Los Angeles».

Por un procedimiento análogo se puede buscar

Este dispositivo de manejo de textos está pensado para usarlo con una sola mano. Se introducen los caracteres —que aparecen en una pantalla de una sola línea— pulsando las teclas. Cada dedo gobierna una tecla, menos el pulgar, que dispone de una segunda para diferentes funciones de control. Puede trabajar con todo el juego de caracteres ASCII. Lleva pilas recargables y suficiente RAM para almacenar unas 1.500 palabras, que se conservan aun cuando apaguemos la máquina. Mediante una *interface* se le puede conectar un magnetófono y cuenta, además, con otra salida RS232.

cualquier cosa que haya en medio del texto sin tener que andar leyéndolo todo.

Otras posibilidades del procesamiento de textos

La idea básica es disponer de una herramienta que permita escribirlo, comprobarlo y pulirlo todo en la pantalla, sin poner nada por escrito mientras no hayamos quedado satisfechos. Pero el procesador también permite controlar la presentación del texto cuando por fin se escribe en un papel.

Para empezar, casi todos los procesadores cuentan con posibilidades básicas tales como: justificación del margen derecho, centrado entre ambos márgenes, numeración automática de páginas, encabezados de las páginas y las notas a pie de página. Dependiendo del tipo de impresora que se use, podrán estar incluidos efectos como el subrayado, los tipos en negrita o cursiva, y el espaciamiento proporcional.

Algunos «paquetes» de procesamiento de textos para micros ofrecen unas prestaciones realmente grandes. Los hay que permiten numerar automáticamente párrafos y subpárrafos, y otros admiten datos procedentes de programas distintos, tales como bases de datos (véase Módulo 34) y hojas de cálculo (véase Módulo 35). La mayoría de los paquetes de procesamiento de textos son capaces de darnos el numero de caracteres de un texto, pero muy pocos el de palabras. Para un periodista podría ser muy útil un paquete capaz de contar palabras que permitiera fijar un tope e hiciera sonar la campanilla cuando llegase a ese número. Un procesador de textos permite la elaboración automática de índices. Se encierran las palabras o frases que se quiere figuren en el índice entre unos símbolos especiales a medida que se escriben (aunque tal vez resulte más fácil hacerlo al final, con la «utilidad» de búsqueda y sustitución), y cuando se imprime el texto se elabora e imprime también el índice.

¿Necesitamos un procesador de textos?

Es muy bonito disponer de un procesador de textos, pero su precio no es bajo. Aunque ya tengamos ordenador, normalmente habrá que comprarlo aparte. A diferencia de los paquetes de base de datos y de hoja de cálculo, cuya utilidad y provecho son obvios, y por tanto fáciles de justificar, el procesamiento de textos es un poco más difícil de evaluar. Ciertos tipos de trabajo, sin embargo, se adecúan especialmente a él, hasta ser sencillamente inviables sin un procesador de textos. Otros, sin embargo, se resuelven mejor con una máquina de escribir corriente.

Para una empresa que ocasionalmente tenga que escribir un ejemplar de carta o de factura no tiene sentido plantearse la adquisición de un procesador de textos: sería una pérdida de tiempo y de dinero. El procesamiento de textos se justifica cuando se tiene un gran volumen de trabajo re-

petitivo o se quieren escribir documentos muy largos, tales como informes, artículos o libros, donde lo más probable es que hagan falta varias revisiones antes de imprimir la versión definitiva.

Como muestra del funcionamiento de un procesador de textos, pensemos en la situación descrita en el Módulo 34, dedicado a las bases de datos: una compañía que envía información sobre un nuevo producto a sus clientes por correo. Existe ya una base de datos en la que figuran el nombre, la dirección, el número de teléfono y otros detalles de cada cliente. Supuesto que tengamos el sistema de base de datos adecuado, podremos crear un fichero en disco con los datos de las personas que deben recibir las cartas: nombre, dirección y tipo de saludo. Con el procesador de textos se escribe una carta-tipo para todos, pero insertando unos símbolos especiales en lugar del nombre, la dirección y el saludo. Esta información se guarda también en disco, pero en un fichero aparte. Después, mediante la utilidad de imprimir del procesador de textos, se le ordena imprimir la carta varias veces, consultando al mismo tiempo el fichero de nombres y direcciones para insertar automáticamente en el lugar adecuado de cada ejemplar de la carta, nombre, dirección y saludo. El resultado será una serie de cartas personalizadas.

Algunos procesadores de textos permiten ir aún más lejos. Supongamos que tenemos una gama variada de productos y nos hemos tomado la molestia, al montar la base de datos, de incluir datos sobre los productos que interesan a los diversos clientes. Si ponemos esta información en el mismo fichero de nombres y direcciones que genera la base de datos para el procesador de textos, nos será factible escribir el marco general de la carta y, en un fichero en disco aparte, unos cuantos párrafos con la descripción de cada producto. Entonces el procesador de textos imprimirá la carta-marco siempre, leyendo luego no sólo los nombres y direcciones, sino también los datos de los productos que interesan al cliente. El procesador

selecciona luego del otro fichero los párrafos pertinentes en los que se describe la gama de nuevos productos, insertándolos también.

Sin embargo, no todos los «paquetes» de procesamiento de textos gozan de esta flexibilidad, y a veces puede resultar difícil generar a partir de la base de datos un fichero que pueda leer el procesador de textos. Por regla general, para este tipo de aplicaciones, lo mejor es adquirir uno de los paquetes «integrados» que hay ahora, constituidos por un conjunto de programas de base de datos, hoja de cálculo y procesamiento de textos, especialmente diseñados para que no haya problemas de compatibilidad.

Los procesadores de textos también están empezando a tener aceptación entre los que tienen que escribir documentos de ejemplar único pero muy extensos. Al que ha de escribir informes técnicos, por ejemplo, la posibilidad de trasladar bloques de texto o de corregir errores mecanográficos ha de resultarle enormemente atractiva. Estos usuarios suelen tener necesidades muy especiales en cuanto a la presentación del texto: párrafos sangrados, espacios en blanco para ilustraciones o diagramas, o problemas de distribución no habituales. Con un procesador de textos se puede incluso sacar un borrador y hacerlo circular entre las personas interesadas para que añadan sus comentarios y correcciones, que luego, una vez tecleados, figurarán en la versión definitiva.

Hardware para procesadores de textos

Supuesto que la elección de ordenador no venga dictada por la existencia de algún otro *software* que nos haga falta, los aspectos más importantes de la selección del *hardware* desde el punto de vista del procesamiento de textos son la pantalla y el teclado.

Trabajar con un procesador de textos supone pasarse muchas horas ante la pantalla, y por eso es importante que sea clara, legible y estable.

Para empezar, es necesario que la pantalla tenga unas mayúsculas y minúsculas legibles,con «bajos» adecuados (los «palos» de letras como la «y» o la «g» han de quedar por debajo de la línea de escritura, porque hay máquinas en las que quedan aplastados). También es necesario contar cuando menos con 24 ó 25 líneas de 80 caracteres, pues todo lo que sea menor que esto termina por resultar incomodísimo.

Otra característica importante de la pantalla es que cuente con una superficie no reflectante y se pueda orientar en cualquier dirección, para que podamos ponerla en la que más cómodo nos resulte. Y debe tener mandos de brillo y de contraste al alcance del operador. El color es importante también: la combinación más usual es caracteres verdes sobre fondo negro, aunque también se puede invertir. Otras combinaciones son: caracteres negros sobre fondo blanco (un contraste bastante brusco, del que los ojos pueden fatigarse) o caracteres ámbar/naranja sobre fondo pardo. Una menos habitual es ámbar sobre púrpura.

El teclado es algo que, al igual que la pantalla, merece la pena estudiar bien antes de efectuar la compra. Debe ser independiente del cuerpo del ordenador, para poder ponerlo donde nos plazca, y la parte principal de la unidad debe tener la distribución y pendiente de las máquinas de escribir corrientes. El «tacto» del teclado de un ordenador es a veces distinto del de una máquina de escribir, y el preferir uno u otro es en buena medida cuestión de gusto personal. Sin embargo, debe haber al menos cuatro teclas para manejar el cursor (arriba, abajo, derecha e izquierda) y es conveniente contar con la posibilidad de obtener otras funciones de uso frecuente pulsando una sola tecla especial, sobre todo para casos de inserciones y borrados. En algunas máquinas, sin embargo, se han llevado las cosas demasiado lejos, poniendo hasta 33 teclas especiales, cada una con dos funciones (según se pulse o no simultáneamente la de activar mayúsculas), con lo que orientarse en ellas es un ejercicio de paciencia y memoria.